Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

A cura di *Marco Minerva*

Il linguaggio di programmazione sviluppato da Microsoft per la scrittura di applicazione compatibili con Windows. Grazie alla sua interfaccia completamente visuale rappresenta, assieme a Delphi, un ottimo punto di partenza per chi vuole avvicinarsi partendo da zero al mondo della programmazione.

Faq sul linguaggio Visual Basic
Decine di risposte alle domande più frequenti sul Visual Basic

Introduzione al linguaggio e panoramica sulle basi della programmazione

1. Visual Basic: scopri quanto è facile programmare
Prime definizione di Visual Basic, primo approccio alla programmazione event driven.

Iniziamo a conoscere Visual Basic

2. I tipi di dati
Introduzione ai tipi di dati alle costanti e alle variabili.

3. Le procedure e le funzioni
Procedure e funzioni sono l'ossatura della programmazione in Visual Basic. Vediamo come inserirle nel nostro progetto.

4. Le strutture iterative
Per poter svolgere un procedimento iterativo Visual Basic utilizza principalmente due istruzioni: For ... Next e Do ... Loop

5. La Casella degli strumenti e la finestra delle Proprietà
Entriamo nell'ambiente di Visual Basic e familiarizziamo con l'interfaccia grafica del programma.

6. Gli eventi fondamentali del mouse
Approfondimento sugli eventi, il perno della programmazione in Visual Basic. Gli eventi del mouse.

7. Gli eventi fondamentali della tastiera
Gli eventi generati da tastiera e la loro applicazione nella programmazione in Visual Basic.

8. Il form, l'elemento centrale di un'applicazione
Il Form (genericamente "finestra") è il punto di maggior contatto fra il programma e l'utente. Vediamo come costruirlo.

Gli esempi delle prime 8 lezioni
Gli esempi di queste prime lezioni raccolto in un file .zip da 9 Kb

I controlli di Visual Basic

9. I controlli di Visual Basic: CommandButton, TextBox e Label
Approfondiamo la programmazione in Visual Basic: descrizione analitica dei primi controlli da utilizzare.

10. I controlli Frame, CheckBox e OptionButton
Continuiamo il nostro esame dei controlli standard di Visual Basic: in questa lezione ci occupiamo dei controlli Frame, CheckBox e OptionButton.

11. I controlli ListBox e ComboBox
Due controlli molto usati in Visual Basic, utili per integrare nei programmi Visual Basic una lista di opzioni selezionabili dall'utente.

12. I controlli ImageBox e PictureBox
Questi due controlli sono utilizzati per inserire nelle proprie applicazioni immagini e fotografie.

13. I controlli DriveListBox, DirListBox e FileListBox
Per accedere ai dischi del sistema e visualizzare le loro informazioni useremo invece questi tre semplici controlli.

14. Il controllo Timer
Con il controllo Timer possiamo controllare la temporizzazione di ogni funzione Visual Basic.

15. La gestione degli errori in Visual Basic
Una funzione importantissima in ogni programma: gestire al meglio gli errori permette di favorire l'usabilità e la funzionalità del programma.

16. Aggiorniamo il programma
Aggiorniamo il nostro programmino creato nella lezione 13 con il codice di gestione degli errori.

17. Aggiungere un controllo OCX al progetto
Con i controlli OCX è possibile arricchire il proprio programma Visual Basic con nuove funzioni e nuovi controlli.

Gli esempi delle ultime 9 lezioni
Gli esempi delle lezioni dalla 9 alla 17 raccolto in un file .zip da 20 Kb

Programmazione pratica: realizziamo un'agenda elettronica

---

18. L'interfaccia
Diamo forma ad una nostra creazione: costruiamo l'interfaccia della nostra prima applicazione in Visual Basic.

19. Attiviamo la navigazione nel database
Ogni agenda dovrà scrivere i dati in una banca dati. Vediamo come creare un database in modo semplice.

20. La funzione di ricerca
Costruiamo l'interfaccia attraverso la quale ricercare nel database i nomi inseriti.

21. La funzione di modifica
Ecco invece come modificare o aggiungere dati nella nostra agenda.

22. L'aggiunta di dati
Ultimi piccoli ritocchi alla nostra agenda: implementiamo la funzione di aggiunta dei dati.

23. L'eliminazione dei dati
Ultima lezione del corso: l'eliminazione dei dati e piccolo consuntivo su ciò che avreste dovuto imparare dalle lezioni.

Gli esempi delle ultime 6 lezioni
Gli esempi delle lezioni dalla 18 alla 23 raccolto in un file .zip da 25 Kb

TORNA SU

FAQ VISUAL BASIC

A cura di *Marco Minerva*

Le Faq al linguaggio Visual Basic vi faranno conoscere tutte le particolarità del linguaggio, vi permetteranno di risolvere i più comuni problemi e vi guideranno nella realizzazione di progetti professionali. Divise in nove "capitoli", sono state pensate come completamento alla guida al Visual Basic.

1. Faq Generali

Domande e risposte sulle caratteristiche generali di Visual Basic: ambiente di lavoro, scrittura del codice, utilizzo del linguaggio.

2. Faq Sintassi Generale

Faq sulla sintassi generale di Visual Basic: comandi, variabili, funzioni.

3. Faq livello Intermedio & API

Faq di livello intermedio e sulle API di Visual Basic

4. Faq Stringhe

Faq sull'utilizzo delle stringhe in Visual Basic

5. Faq Controlli e Stampa

Faq sui controlli per l'interfaccia utente e sulle funzioni di stampa

Altre FAQ saranno pubblicate nel corso delle prossime settimane

TORNA SU

FAQ VISUAL BASIC

Faq Generali

- Quali sono le differenze tra le versioni di VB Learning, Professional ed Enterprise? **[risposta]**
- Esistono versioni gratuite di Visual Basic? **[risposta]**
- Che cosa sono i Service Pack? Quando conviene installarli? **[risposta]**
- Ho notato che in un progetto posso inserire oggetti di vario tipo, come Form, Moduli e Moduli di classe. Che differenza c'è tra di essi? **[risposta]**
- Che differenza c'è tra form e form MDI? **[risposta]**
- Quando vado sul comando "Inserisci form", mi appare una finestra in cui sono presenti diversi modelli predefiniti; come posso aggiungere i miei modelli? **[risposta]**
- Come si compila un programma? **[risposta]**
- Ho notato che, nel menu Esegui, sono disponibili due comandi per avviare un progetto, "Avvia" ed "Avvia con compilazione" completa; che differenza c'è tra i due? **[risposta]**
- Come si cambia l'icona di un programma? **[risposta]**
- Come si cambia il titolo di un'applicazione? **[risposta]**
- I file che compongono la mia applicazione VB sono sparsi in varie cartelle; volendo metterli tutti in un'unica directory, ho provato semplicemente a spostarli, ma a quel punto non riesco più ad aprire il progetto, ma ottengo vari messaggi che mi informano che certi file non vengono trovati. Come mai? **[risposta]**
- Che cosa significa "indentare"? **[risposta]**
- Ho notato che il mio programma rimane in esecuzione anche quando chiudo la finestra principale. Come mai? **[risposta]**
- È possibile fare in modo che un'applicazione VB accetti argomenti dalla riga di comando? Come li posso recuperare? **[risposta]**
- La mia applicazione accetta dei parametri dalla riga di comando. C'è un modo semplice per fare qualche prova con questi parametri, senza dover ogni volta compilare il programma? **[risposta]**
- Sento spesso dire che l'ambiente di sviluppo di Visual Basic utilizza la tecnologia IntelliSense. Che cosa si intende con questo termine? **[risposta]**
- Come posso recuperare la lista dei Fonts installati nel sistema? **[risposta]**
- Con Visual Basic è possibile realizzare giochi? **[risposta]**
- Che differenza c'è tra codice nativo e p-code? **[risposta]**
- Che differenza c'è tra il passaggio di variabili per valore e il passaggio per riferimento? **[risposta]**

FAQ VISUAL BASIC

Faq Sintassi Generale

- Che cosa significa il comando Option Explicit, che talvolta è inserito nella sezione Dichiarazioni di un modulo, di un form, ecc.? **[risposta]**
- Che cosa significa il comando Option Compare Text, che talvolta è inserito nella sezione Dichiarazioni di un modulo, di un form, ecc.? **[risposta]**
- Come si usa la funzione Format? **[risposta]**
- A cosa serve la proprietà TabIndex? **[risposta]**
- Come si usa l'evento Validate? **[risposta]**
- Utilizzando la funzione Rnd ottengo sempre la stessa sequenza di numeri. Come mai? **[risposta]**
- Quando conviene utilizzare il costrutto Select Case invece di If… Then… Else? **[risposta]**
- Come funziona l'istruzione End? **[risposta]**
- Che differenza c'è tra un ciclo realizzato con While e uno realizzato con Until? **[risposta]**
- Posso impedire che il mio programma venga visualizzato nella barra delle applicazioni? **[risposta]**
- Come si crea un tipo di dato personalizzato? **[risposta]**
- Che cos'è un'enumerazione? **[risposta]**
- E' possibile convertire un dato da un tipo ad un altro? **[risposta]**
- Dato il nome completo di un file, come posso recuperarne solo il nome o il percorso? **[risposta]**
- Nella mia applicazione posso visualizzare un menu contestuale, come quello utilizzato praticamente da tutte le applicazioni per Windows? Come? **[risposta]**
- Se io non specifico il tipo di dato di una variabile, Visual Basic la definisce come Variant. Se io ho una lunga serie di variabili dello stesso tipo, c'e' un modo per evitare di specificare per ognuna il tipo di dati che conterrà? **[risposta]**
- Se ho una serie di variabili dello stesso tipo, scritte tutte sulla stessa riga, devo specificare il tipo per ognuna, oppure posso metterlo solo alla fine, come accade ad esempio in Pascal? **[risposta]**
- Come si lancia da VB un altro programma EXE? **[risposta]**
- E' possibile effettuare calcoli con le date? **[risposta]**
- Vorrei fare in modo che, premendo il pulsante X sulla barra del titolo della mia applicazione, compaia un finestra che chiede se si è sicuri di voler uscire e, in caso negativo, annulli la chiusura del programma. E' possibile fare una

cosa del genere? [risposta]

- Se voglio modificare una serie di proprietà di un oggetto, posso evitare di dover scrivere ogni volta il nome dell'oggetto stesso? [risposta]
- Che differenza c'è tra matrici statiche e matrici dinamiche? Come si ridimensiona una matrice? [risposta]
- Come si recupera la lunghezza di un file? [risposta]
- Che cosa significa la parola chiave Static? [risposta]
- Che cosa significa la parola chiave Optional? [risposta]
- Posso controllare se il valore immesso è un numero oppure una data valida? [risposta]

[S o m m a r i o]

TORNA SU

FAQ VISUAL BASIC

Faq Intermedio & API

- A che cosa è dovuto l'errore 429? [risposta]
- Posso sapere se la mia applicazione è già in esecuzione? [risposta]
- Posso sapere se una determinata applicazione è in esecuzione? [risposta]
- Invece di preoccuparmi di scaricare tutti i form, per uscire dal mio programma non posso più semplicemente utilizzare l'istruzione End? [risposta]
- Qual è il modo più efficiente per scaricare tutti i form del mio programma che sono ancora in esecuzione? [risposta]
- Nel mio programma eseguo una lunga serie di operazioni, durante la quale il sistema sembra "bloccato". Posso ovviare a questa situazione? [risposta]
- Come si recupera la risoluzione dello schermo? [risposta]
- Ho un file il cui nome è in formato breve 8+3; posso recuperare il corrispondente nome lungo? [risposta]
- Posso convertire il nome di un file in formato breve 8+3? [risposta]
- Che cosa sono le API di Windows? [risposta]
- Quando si parla di API di Windows, spesso si parla anche di handle; di cosa si tratta? [risposta]
- Posso scrivere nel Registro di configurazione di Windows? [risposta]
- Vorrei che la mia applicazione venisse eseguita ad ogni di Windows. Come posso fare? [risposta]
- Come posso recuperare il percorso della directory di Windows e quello della directory System? [risposta]
- Qual è il modo migliore per controllare l'esistenza di un file o di una cartella? [risposta]
- Che cos'è e a cosa serve il Visualizzatore API? [risposta]
- Voglio realizzare un'applicazione multilingua: qual è il modo migliore per farlo? [risposta]
- Che cos'è la compilazione condizionale? [risposta]
- Dalla mia applicazione posso aprire il browser Internet e visualizzare una pagina da me impostata? [risposta]
- Dalla mia applicazione posso aprire il client di posta elettronica per mandare un messaggio all'indirizzo specificato? [risposta]
- C'è un modo per recuperare lo spazio su disco occupato e quello libero? [risposta]

- Con Visual Basic si possono realizzare Screen Saver? [risposta]
- E' possibile riprodurre un file audio da un'applicazione Visual Basic? [risposta]
- Volendo modificare la proprietà MouseIcon di un form, ho notato che è possibile selezionare solo file .ICO e .CUR. C'è un modo che mi permetta di utilizzare anche puntatori animati (.ANI)? [risposta]
- A cosa serve il programma RegSvr32.exe? [risposta]
- Posso eseguire comandi MS-DOS da Visual Basic? [risposta]
- Esiste una funzione che mi permette di recuperare il percorso delle cartelle di sistema (quella dei Documenti, quella dei Cookies, ecc.)? [risposta]
- Come posso aprire un file con il programma associato? [risposta]
- Come si creano i file della Guida? [risposta]
- Nel mio programma vorrei utilizzare la finestra "Sfoglia per cartelle" tipica di Windows. Come posso fare? [risposta]
- Posso mettere la mia applicazione sempre in primo piano? [risposta]
- E' possibile ridimensionare in modo automatico i controlli inseriti in un form al variare della risoluzione dello schermo? [risposta]
- Come si calcola il giorno di Pasqua? [risposta]

[S o m m a r i o]

TORNA SU

FAQ VISUAL BASIC

Faq Stringhe

- E' possibile convertire il testo in formato tutto maiuscolo o tutto minuscolo? [risposta]
- Ho notato, leggendo anche il codice VB che si può scaricare da Internet, che alcune funzioni terminano con il carattere $ (come Left$, Right$, Mid$, UCase$, LCase$, ecc.); tuttavia, se rimuovo questo carattere, il codice viene eseguito ugualmente. Qual e' dunque il significato del carattere $? [risposta]
- Vorrei visualizzare i separatori delle migliaia in un numero. C'è un modo per farlo in modo efficiente? [risposta]
- Ho una stringa in cui alcuni valori sono separati da un ben preciso carattere; c'è un modo efficiente per considerare separatamente ogni valore? [risposta]
- Come posso recuperare l'intero contenuto di un file di testo? [risposta]
- Posso controllare se una stringa contiene al suo interno una determinata sequenza di caratteri? [risposta]
- Qual e' il modo migliore per sostituire all'interno di una stringa tutte le occorrenze di un carattere con un'altra? [risposta]

[Sommario]

TORNA SU

FAQ VISUAL BASIC

Faq Controlli e Stampa

- Come posso impedire la digitazione in una casella di testo? **[risposta]**
- Come posso fare in modo che in una casella di testo vengano digitati solo determinati caratteri? **[risposta]**
- E' possibile fare in modo che il contenuto di una Label si estenda automaticamente su più righe? **[risposta]**
- Utilizzo l'oggetto Printer per inviare alcuni dati alla stampante, tuttavia ho notato che la stampa non viene avviata finché non si esce dall'applicazione. Come mai? **[risposta]**
- Nelle normali applicazioni per Windows le etichette, i pulsanti e i comandi di menu hanno di solito un carattere sottolineato, che, se premuto in combinazione con il tasto ALT, consente di accedere in modo rapido alla funzione corrispondente. E' possibile fare la stessa cosa in VB? **[risposta]**
- A volte mi capita che, aggiornando il contenuto di una Label, questo non venga visualizzato immediatamente. Posso risolvere questo problema? **[risposta]**
- Con l'oggetto Printer, come posso specificare la posizione in cui desidero stampare? Posso stampare del testo allineato a destra oppure al centro? **[risposta]**
- Come si crea una matrice di controlli in fase di progettazione? Una volta creata, come si può accedere alle proprietà dei singoli controlli che fanno parte della matrice? **[risposta]**
- Come si crea una matrice di controlli a runtime? **[risposta]**
- Posso fare in modo che gli elementi contenuti in una ListBox vengano ordinati alfabeticamente? **[risposta]**
- In alcuni programmi, agendo su appositi pulsanti è possibile cambiare l'ordine degli elementi presentati in una ListBox. E' possibile fare la stessa cosa in VB? **[risposta]**
- Ho notato che, se nella proprietà Caption di un oggetto metto una &, il carattere successivo che digito appare sottolineato; e se invece voglio proprio visualizzare la & come faccio? **[risposta]**
- E' possibile visualizzare il testo digitato in una TextBox in modalità password? **[risposta]**
- E' possibile impostare la lunghezza massima del testo che si può immettere in una TextBox? **[risposta]**
- Nel mio programma devo visualizzare una semplice immagine. E' meglio usare il controllo Image oppure il PictureBox? Perché? **[risposta]**
- Se in una casella di testo premo il tasto Invio si sente un beep. E' possibile evitarlo? **[risposta]**
- Ho notato che i controlli Image e PictureBox consentono di visualizzare

immagini in formato JPEG, ma solo di salvare in BMP. Non è possibile salvare un'immagine in formato JPEG? [risposta]

- In un form devo visualizzare un'immagine più grande dell'area visibile. E' possibile creare delle barre di scorrimento per visualizzarla? [risposta]
- Se nella proprietà Text di una TextBox inserisco il carattere di ritorno a capo (vbCr), nella casella appare una linea verticale e il testo continua ad essere scritto sulla stessa riga, mentre io vorrei che andasse a capo. Come mai? [risposta]
- Posso fare in modo che, quando una TextBox riceve lo stato attivo, venga automaticamente selezionato tutto il testo contenuto al suo interno? [risposta]
- Come posso recuperare il testo selezionato in una TextBox? [risposta]
- E' possibile fare in modo che, facendo clic con il tasto destro del mouse su una casella di testo, venga visualizzato un menu contestuale diverso da quello predefinito? [risposta]
- E' possibile ridimensionare un'immagine da VB? [risposta]
- Nella mia applicazione vorrei inserire dei menu in stile Office, ovvero con le icone accanto ai vari comandi. E' possibile? [risposta]

[S o m m a r i o]

TORNA SU

GUIDA AL VISUAL BASIC

LEZIONE 1: *Scopri quanto è facile programmare*

Al giorno d'oggi sono disponibili numerosi linguaggi e tool per sviluppare applicazioni, dal C a Java, a Delphi; di questo gruppo da parte anche **Visual Basic** , un linguaggio di programmazione che alcuni ritengono limitato e troppo "semplicistico".

Forse questa definizione si poteva applicare alle prime versioni del software: a partire dalla release 4, ma soprattutto con le versioni 5 e 6, Visual Basic è diventato un linguaggio di programmazione di ottimo livello, che consente di realizzare programmi praticamente di ogni tipo, dall'editor di testo al server web.

Le caratteristiche che fanno di Visual Basic un linguaggio di programmazione estremamente versatile e facile da usare sono due: le funzioni di progettazione dell'interfaccia completamente visuali; il linguaggio di tipo **event-driven**. L'ambiente di sviluppo visuale consente di essere produttivi fin da subito.

Visualizza l'interfaccia

Non appena si avvia VB, nell'area centrale, si può osservare una finestra, il **form**, che rappresenta la finestra della nostra applicazione. Per inserire elementi all'interno del form (i cosiddetti controlli), quali pulsanti, caselle di testo, etichette, è sufficiente selezionarli all'interno della Casella degli strumenti e trascinarli sul form stesso: il controllo selezionato verrà posizionato nel punto esatto che si è deciso. Altrettanto facilmente è possibile modificare la posizione e la dimensione di un controllo semplicemente utilizzando il mouse.

L'altra caratteristica di Visual Basic cui si è accennato prima è quella di essere un linguaggio **event-driven**. Con questo termine si intende che l'elemento che sta alla base del linguaggio è l'evento, cioè, più in generale, l'azione: un evento è il clic dell'utente su un pulsante, la digitazione in una casella di testo, la selezione di un comando di menu, ma anche il cambiamento della risoluzione, l'aggiunta di una periferica al sistema, ecc. Come vedremo meglio più avanti, gli oggetti inseriti in un form Visual Basic sono in grado di riconoscere in automatico gli eventi più comuni, senza bisogno che il programmatore si preoccupi, ad esempio, di stabilire quando l'utente fa clic su un pulsante, seleziona un elemento da una lista, ecc.

Grazie a queste (e a molte altre) peculiarità, Visual Basic è un linguaggio di programmazione facile da usare ma, nello stesso tempo, potente e flessibile. Lo scopo di questo corso è di illustrare le caratteristiche principali di Visual Basic, allo scopo di fornire una base sulla quale sia possibile fondare uno studio più approfondito del linguaggio. Nelle prime lezioni verranno illustrati alcuni concetti che sono alla base del linguaggio (differenza tra costanti e variabili, funzioni e procedure, eventi, ecc.) e, nello stesso tempo, verrà spiegato dettagliatamente come iniziare ad utilizzare VB.

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 2: *I tipi di dati*

Prima di entrare nel "vivo" di questo corso e di iniziare a lavorare con Visual Basic, è necessario spendere qualche parola su alcuni concetti molto importanti che stanno alla base di ogni linguaggio di programmazione; questa lezione può sembrare noiosa, ma contiene delle informazioni fondamentali, senza conoscere le quali è impossibile procedere con lo studio di VB.

Innanzi tutto, vediamo di capire cosa sono le *costanti* e le *variabili*. E' possibile pensare ad esse come a contenitori in cui si trovano delle informazioni, cioè dei valori; più precisamente, costanti e variabili sono riferimenti a locazioni di memoria in cui sono salvati determinati valori. Non ci interessa sapere qual è l'esatto indirizzo della memoria che contiene questi valori: è Visual Basic che si occupa di andare a recuperare nella memoria il valore associato alla variabile o alla costante che stiamo utilizzando. La differenza tra costanti e variabili è questa: le costanti, come dice il nome stesso, una volta impostate non sono più modificabili, mentre le variabili (anche in questo caso il nome è illuminante) possono essere modificate ogni volta che si desidera. Ad esempio, se io creo (tra poco vedremo come) una costante di nome **Pippo** e imposto il suo valore su 10, in seguito non posso modificarla, quindi tale costante varrà 10 per tutta l'esecuzione del programma. Se, invece, ho una variabile **Pluto**, posso modificare il suo valore in ogni momento, quindi posso inizialmente assegnarli il valore 4, poi 9, poi 3, e così via.

Un altro punto nodale in qualsiasi linguaggio di programmazione è la distinzione dei **tipi di dato**. Come è facile intuire, un programma lavora con dati di tipo diverso, cioè stringhe (ovvero sequenze di caratteri) e numeri; questi ultimi, poi, si dividono ulteriormente a seconda che siano numeri interi, decimali, che indichino valute, ecc. Questa distinzione è molto importante, perché ogni tipo di dato ha una dimensione (cioè un'occupazione in memoria) diversa: ad esempio, un numero intero occupa meno memoria di un numero decimale a precisione doppia. Tali particolari possono sembrare delle sottigliezze, però quando si sviluppano applicazioni di una certa complessità essi vengono ad assumere un'importanza rilevante. Vediamo ora i tipi di dati fondamentali in Visual Basic, ricordando che nella Guida in linea del linguaggio è possibile trovare altre informazioni su questo argomento:

Tipo di dato	Dimensione in memoria	Intervallo
Boolean	2 byte	True (-1) o False (0)
Integer (intero)	2 byte	Da -32.768 a 32.767
Long (intero lungo)	4 byte	Da -2.147.483.648 a 2.147.483.6477

Single (virgola mobile a precisione semplice)	4 byte	Da -3,402823E38 a -1,401298E-45 per valori negativi; da 1,401298E-45 a 3,402823E38 per valori positivi
Double (virgola mobile a precisione doppia)	8 byte	Da -1,79769313486232E308 a -4,94065645841247E-324 per valori negativi; da 4,94065645841247E-324 a 1,79769313486232E308 per valori positivi
String	10 byte + lunghezza stringa (10 byte + numero caratteri)	Da 0 a circa 2 miliardi

Detto questo, possiamo ritornare al discorso con cui abbiamo aperto la lezione e analizzare la dichiarazioni di costanti e variabili in VB. Le costanti si dichiarano in questo modo:

Const [As <Tipo>] =

Const è una parola chiave riservata di VB che si usa per definire una costante. è il nome che si sceglie di attribuire alla costante. Nella scelta dei nomi (sia delle costanti, delle variabili, ma anche delle procedure, delle funzioni e dei controlli, che vedremo più avanti), è necessario seguire alcune regole. I nomi non devono essere più lunghi di 40 caratteri, non posso iniziare con un numero né contenere spazi e caratteri come ?, !, :, ;, . e ,.
Visual Basic, a differenza di altri linguaggi come il C o Java, non fa differenza tra maiuscole e minuscole.
As <Tipo> è un parametro opzionale che indica il tipo di dato contenuto nella costante; se non viene specificato, il compilatore lo determinerà sulla base del valore assegnato alla costante stessa. è il valore vero e proprio della costante.
Ecco alcuni esempi di dichiarazioni di costanti:

Const PI = 3.14 Const Nome As String = "Marco"

Una sintassi analoga è quella che permette di dichiarare le variabili:

Dim <nome> [As <Tipo>]

In questo caso si usa la parola chiave Dim per indicare al compilatore che quella che si sta per definire è una variabile. Le convenzioni per il nome sono le stesse che sono state accennate a proposito delle costanti. Anche per le variabili il parametro **As <Tipo>** è opzionale: se non viene specificato, la variabile verrà dichiarata di tipo Variant, un particolare tipo che può contenere dati di tutti i tipi. E' sconsigliabile definire variabili di tipo Variant, se non espressamente necessario, dal momento che questo tipo di dato occupa molta memoria. Ecco alcuni esempi di dichiarazioni di variabili.

Dim Utenti As Integer Dim Nome As String, Cognome As String

Per maggiori informazioni sugli argomenti trattati in questa lezione, cercare l'argomento Riepilogo dei tipi di dati nella Guida in linea di Visual Basic.

Lezione successiva

[Sommario]

TORNA SU

GUIDA AL VISUAL BASIC

LEZIONE 3: ***Le procedure e le funzioni***

Dopo aver parlato di tipi di dati, costanti e variabili, e prima di dedicarci alla programmazione vera e propria, dobbiamo ancora illustrare brevemente le procedure e le funzioni. In entrambi i casi si tratti di una sorta di "raggruppamento" di istruzioni che svolgono un'operazione comune. Come vedremo meglio più avanti, praticamente tutto il codice di un programma Visual Basic è contenuto all'interno di funzioni e procedure (chiamate genericamente **routine**). La differenza fondamentale tra procedure e funzioni è che le seconde possono restituire dei valori, ad esempio il risultato di un'elaborazione oppure un valore di ritorno che determina se la routine ha avuto successo, mentre le procedure no. Iniziamo a vedere la dichiarazioni di una procedura:

Sub <nome>([Parametro As <Tipo>, …])
…
End Sub

Tutte le dichiarazioni di procedura iniziano con la parole chiave Sub. Segue il nome della routine, che deve rispettare le convenzione analizzate precedente a proposito delle costanti. Il nome deve essere seguito da parentesi, al cui interno è possibile inserire i parametri (opzionali) richiesti della procedura; non c'è limite al numero di parametri che si possono definire. Tali parametri possono essere visti come variabili (ritorneremo tra poco su questo punto). **End Sub** sono parole riservate di VB che indicano la fine di una procedura. Vediamo ora un esempio di procedura, anche per illustrare meglio l'utilizzo dei parametri. Supponiamo di dover calcolare l'area di un cerchio: la formula è sempre la stessa, quello che cambia è solo la misura del raggio. Per tale motivo, invece di riscrivere ogni volta la formula, possiamo scrivere una procedura che richieda come parametro proprio la lunghezza del raggio:

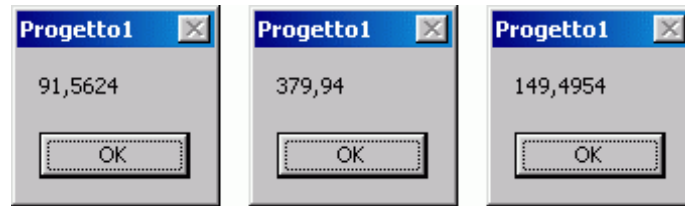
Sub AreaCerchio(Raggio As Double)
…
End Sub

Supponiamo ancora di voler scrivere un programma che chiede all'utente la lunghezza del raggio e sulla base di questa calcola l'area del cerchio. Dopo aver definito la procedura come sopra descritto, ci basterà richiamarla passandogli come argomento la lunghezza del raggio; ad esempio:

AreaCerchio 5.4
AreaCerchio 11
AreaCerchio 6.9

Queste sono tre **chiamate** alla procedura con parametri diversi. Nel primo caso, Raggio varrà 5.4, nel secondo 11 e nel terzo 6.9. Ecco quindi come potrebbe risultare la procedura AreaCerchio completa:

*Sub AreaCerchio(Raggio As Double) MsgBox Raggio * Raggio * 3.14 End Sub*

In questo esempio è stata usata la funzione MsgBox, che visualizza un messaggio in una finestra di dialogo e attende che l'utente prema un tasto. A questo punto, utilizzando le tre chiamate sopra definite, otterremo questi risultati:

Passiamo ora ad analizzare le funzione, osservando che per esse vale la maggior parte delle considerazioni che già si sono fatte per le procedure. La dichiarazione di una funzione è questa:

Function <nome>([Parametro As <Tipo>, …]) [As <Tipo>]
…
End Function

Come si vede, in questo caso invece della parola chiave Sub si usa Function. La cosa nuova, cui si è già accennato, è che le funzioni possono restituire un valore. Nella dichiarazione, infatti, possiamo notare che, dopo l'elenco (opzionale) dei parametri c'è un ulteriore argomento opzionale, ancora una volta As <Tipo>: esso indica il tipo di dato restituito dalla funzione. Come si è già visto per le variabili, se non viene specificato tale parametro il valore restituito sarà di tipo Variant. Riprendiamo l'esempio di prima e trasformiamo la procedura AreaCerchio in una funzione:

```
Function AreaCerchio(Raggio As Double) As Double
AreaCerchio = Raggio * Raggio * 3.14
End Function
```

Quando si richiama questa funzione, AreaCerchio contiene il valore dell'area del cerchio. Vediamo ora come si utilizzano le funzioni, basandoci come sempre sull'esempio.

```
Dim Area1 As Double, Area2 As Double, Area3 As Double
Area1 = AreaCerchio(5.4) 'Area1 vale 91,5624
Area2 = AreaCerchio(11) 'Area2 vale 379,94
Area3 = AreaCerchio(6.9) 'Area3 vale 149,4954
```

Innanzi tutto sono state dichiarate tre variabili, **Area1, Area2, Area3**, che dovranno contenere i valori dell'area. Ad esse è stato poi assegnato il valore restituito dalla funzione AreaCerchio. Di fianco ad ogni istruzione è stato posto un commento; per inserire un commento in VB è necessario digitare un apice ('): tutto quello che verrà scritto sulla stessa riga a destra dell'apice verrà considerato, appunto, come un commento, pertanto non verrà eseguito. Se adesso noi scrivessimo queste tre istruzioni:

```
MsgBox Area1
MsgBox Area2
MsgBox Area3
```

otterremo lo stesso risultato che è stato mostrato prima, cioè le tre finestre di messaggio contenenti rispettivamente 91,5624, 379,94 e 149,4954.

Prima di concludere questa panoramica, dobbiamo ancora presentare alcune istruzioni di VB che sono utilizzate praticamente in tutti i programmi: le **strutture iterative**, che saranno l'argomento della prossima lezione.

Lezione successiva
[Sommario]

GUIDA AL VISUAL BASIC

LEZIONE 4: *Le strutture iterative*

Le strutture iterative consentono di eseguire più volte una determinata porzione di codice. Le strutture più utilizzate in VB sono due: **For… Next** e **Do… Loop**. La prima è senza dubbio la più utilizzata; la sua sintassi è:

```
For <Contatore> = Inizio To Fine [Step Incremento]
…
Next [<Contatore>]
```

<*Contatore*> è una variabile che deve contenere valori di tipo numerico (quindi può essere Integer, Long, Single, Double, ecc.), così come numerici devono essere i valori di *Inizio, Fine e Incremento*. La parola chiave **Step** è facoltativa, se non viene specificata **Incremento** viene automaticamente impostato a 1. Quando si entra in un ciclo For, la variabile *Contatore* assume il valore specificato in Inizio; subito dopo viene verificato se *Contatore* è maggiore dell'argomento Fine: in tal caso il ciclo termina (analogamente, se Incremento è negativo, viene verificato se *Contatore* è minore dell'argomento Fine). Se, invece, *Contatore* è minore o uguale a Fine (oppure è maggiore o uguale, nel caso che Incremento sia negativo), vengono eseguite le istruzioni all'interno del ciclo e, infine, *Contatore* viene incrementato del valore di Incremento. Queste operazioni vengono fino a quando il valore di *Contatore* diventa maggiore del valore di Incremento (oppure minore se Incremento è negativo). Per uscire dal ciclo prima che si verifichino le condizioni di fine descritte sopra è possibile usare l'istruzione Exit For; con la quale si passa subito ad eseguire le istruzioni successive al ciclo.

Vediamo un semplice esempio di utilizzo di un ciclo For per determinare se un numero è primo. Vogliamo creare una routine, come abbiamo imparato a fare nella lezione precedente. Ecco il codice:

```
Private Sub Primo(N As Long)
 Dim I As Long
   For I = 2 To Sqr(N)
     If N Mod I = 0 Then
       MsgBox "Il numero non è primo."
     Exit For
   End If
 Next I
End Sub
```

Questa procedura prende in ingresso un numero N, di tipo Long; viene poi fatto un ciclo For da 2 alla radice quadrata di N (Sqr è proprio la funzione VB che calcola la radice quadrata di un numero). Ad ogni iterazione il numero N viene diviso per il valore di I (quindi 2, 3… Sqr(N)); si utilizza l'operatore Mod, che restituisce il resto della divisione: se è 0, significa che il numero è divisibile per quel valore di I, quindi non è primo. L'altra struttura iterativa cui abbiamo accennato è quella Do… Loop; di solito viene utilizzata quando non si sa a priori per quante volte è necessario eseguire un certo blocco di codice. Questo costrutto si può presentare in due forme; la più comune è la seguente:

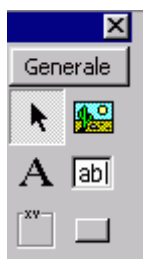
```
Do While Condizione
```

…
Loop

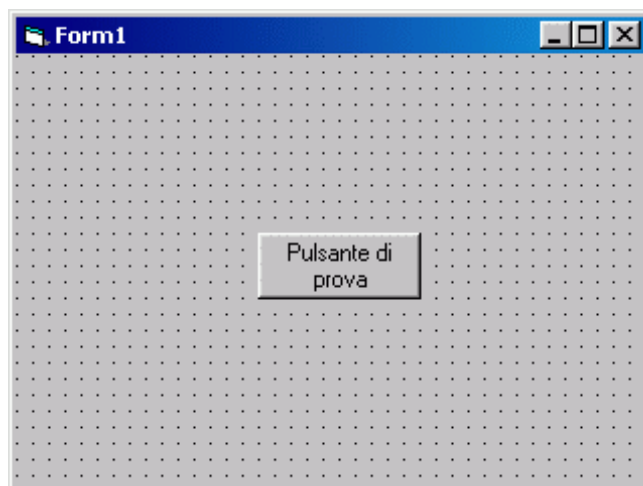
L'esecuzione di questa struttura prevede innanzitutto la verifica della Condizione, che deve restituire un valore di tipo booleano. Se risulta False, tutte le istruzioni del ciclo vengono ignorate, se invece risulta True, le istruzioni vengono eseguite e, di seguito, la condizione viene nuovamente verificata, e così via, finché Condizione risulta False. E' facile intuire che se la condizione risulta subito False, le istruzioni non verranno mai eseguite. L'altra forma del Do… Loop, invece, permette di eseguire le istruzioni e di verificare la Condizione al termine di ciascuna esecuzione. In questo modo le istruzioni vengono eseguite almeno una volta:

Do
…
Loop While Condizione

Con questa lezione termina la panoramica sulle nozioni fondamentali che è necessario conoscere per iniziare a programmare in Visual Basic. Ora possiamo finalmente iniziare a scoprire cosa abbiamo a disposizione per creare un'applicazione: vediamo innanzi tutto cosa sono e come si usano la **Casella degli strumenti** e la finestra delle **Proprietà**.

Lezione successiva
[Sommario]
TORNA SU

HTML.IT SHOP| CORSI IN AULA| FREE INTERNET| WEBTOOL BLOG| CREA| DOWNLOAD| FORUM| LIBRI| NEWSLETTER ADSL| VOIP| HOSTING ASP| B2B| FLASH-MX| FONT| GIF| LINUX| NEWS| PHP| PRO| PROGRAMMAZIONE| SICUREZZA

cerca vai

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 5: *La Casella degli strumenti e la finestra delle Proprietà*

Tutti i **controlli** che possono essere inseriti un *form* Visual Basic sono visualizzati sotto forma di icona nella Casella degli strumenti, una barra laterale che nell'impostazione predefinita è visualizzata sulla sinistra; ognuna di queste icone rappresenta un diverso controllo inseribile. Per provare ad inserire un controllo nel form, fare doppio clic su un'icona contenuta nella casella degli strumenti: l'elemento selezionato verrà visualizzato al centro della finestra. Se ora si seleziona l'elemento appena aggiunto con il mouse, possiamo vedere che la finestra delle *Proprietà* (di solito sulla destra) conterrà tutte le proprietà dell'oggetto stesso.

Modificando queste proprietà è possibile cambiare l'aspetto e le caratteristiche del controllo. Facciamo subito una prova. Fate doppio clic sull'icona della Casella degli strumenti che rappresenta un pulsante; la potete identificare facilmente perché, tenendo il mouse fermo su di essa per qualche istante, verrà visualizzato un tooltip contenente il messaggio *CommandButton*. Un pulsante verrà aggiunto al centro del form; sopra di esso è scritto **Command1**: è, questa, la caption del pulsante. Per modificarla, selezionate il pulsante e fate clic a destra della scritta Caption, visualizzata nella finestra delle Proprietà.

Ora potete digitare la nuova etichetta del pulsante, che verrà modificata

durante la digitazione. Ad esempio, provate a scrivere *Pulsante di prova*. Se avete seguito correttamente questi passaggi, dovreste ottenere un risultato simile a quello mostrato nella figura qui a lato. Se non avete capito qualcosa, oppure se desiderate scaricare questo piccolo esempio, potete fare clic qui per prelevarlo.

C'è anche un altro modo per modificare le proprietà di un controllo inserito in un form: è possibile cambiare la proprietà da codice. In Visual Basic, per accedere da codice alle proprietà di un controllo, è necessario scrivere il nome del controllo stesso (che è il primo valore elencato nella finestra Proprietà), seguito da un punto (.) e dal nome della proprietà che si vuole modificare; poi si deve digitare un uguale (=) e specificare finalmente il nuovo valore della proprietà. Ritornando al nostro esempio, per modificare la caption del pulsante da codice l'istruzione da scrivere è questa:

Command1.Caption = "Pulsante di prova"

Notate che, dopo aver digitato il punto, verrà visualizzato un elenco delle proprietà e dei metodi del controllo; mentre le proprietà consentono di impostare le caratteristiche dell'oggetto, i metodi sono azioni che il controllo può eseguire. Ad esempio, utilizzando il metodo Move, possiamo spostare il controllo in una qualsiasi posizione del form:

Command1.Move 0, 0

Questa istruzione sposta il pulsante nell'angolo in alto a sinistra del form. Come vedremo meglio nelle prossime lezione, ogni controllo dispone di proprietà e di metodi specifici.

Abbiamo così dato un rapido sguardo alla *Casella degli strumenti* e alla finestra delle *Proprietà* di Visual Basic; la descrizione di tutte le proprietà disponibili esula dagli scopi di questo corso. Chi volesse approfondire tale punto può selezionare la proprietà di cui vuole conoscere maggiori informazioni e premere il tasto F1: verrà così visualizzata la Guida in linea di Visual Basic relativa alla proprietà selezionata.

Lezione successiva

[Sommario]

TORNA SU

HTML.it PROGRAMMAZIONE

HTML.IT SHOP| CORSI IN AULA| FREE INTERNET| WEBTOOL BLOG| CREA| DOWNLOAD| FORUM| LIBRI| NEWSLETTER ADSL| VOIP| HOSTING ASP| B2B| FLASH-MX| FONT| GIF| LINUX| NEWS| PHP| PRO| PROGRAMMAZIONE| SICUREZZA

cerca vai

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 6: *Gli eventi fondamentali del mouse*

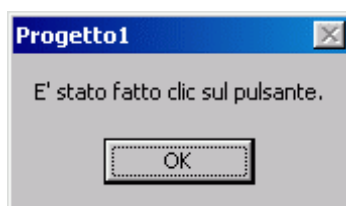
Ricapitoliamo quanto abbiamo visto finora: la dichiarazione di costanti e variabili, la differenza tra funzioni e procedure, la Casella degli strumenti e la finestra delle Proprietà di Visual Basic.

Per **concludere** il nostro tour introduttivo e passare finalmente ad analizzare il linguaggio vero e proprio, dobbiamo ancora spendere qualche parola sugli eventi, cui si e' accennato nella prima lezione, quando abbiamo definito Visual Basic un linguaggio event-driven. Gli **eventi**, cioè le azioni che vengono scatenate dall'utente oppure che sono generate da particolari condizioni (l'impostazione di un timer, la chiusura di Windows, ecc.) sono il **perno** attorno a cui ruota tutta l'attività di programmazione con VB. In questa lezione analizzeremo alcuni tra gli event più comuni, per poi vedere nelle lezioni successive come questi possano essere utilizzati.

Cominciamo con gli eventi principali che si possono generare con il mouse; ci occuperemo degli eventi della tastiera nella lezione successiva. Essi sono fondamentalmente 5: *Click, DlbClick, MouseDown, MouseUp e MouseMove*. L'evento **Click** si verifica quando un utente fa clic con il tasto sinistro del mouse (o destro, se è mancino) sopra un controllo, come un pulsante, una casella di testo, ecc. L'evento **DlbClick**, invece, viene scatenato quando si fa doppio clic sul controllo, per convenzione viene usato quando si vuole sveltire un'operazione, facendo compiere all'utente contemporaneamente l'azione di scelta e quella di conferma. E' importante notare che quando l'utente effettua un doppio clic su un controllo, viene eseguito il codice dell'evento Click e poi quello dell'evento **DblClick**. Facciamo subito una prova per verificare quanto si è detto. Inserite un controllo pulsante (CommandButton) nel form, come abbiamo visto nella lezione precedente. Ora fate doppio clic su di esso; verrà visualizzata questa routine:

Private Sub Command1_Click()
End Sub

E' qui che va inserito il codice che si vuole eseguire quando un utente fa clic sul pulsante. Ad esempio, scrivete:

MsgBox "E' stato fatto clic sul pulsante."

Abbiamo già incontrato in un esempio il comando MsgBox, nelle lezione 3 , quando abbiamo parlato di procedure e funzioni. Ora dobbiamo avviare l'applicazione; per fare questo ci sono tre modi: premete il tasto **F5**; fate clic

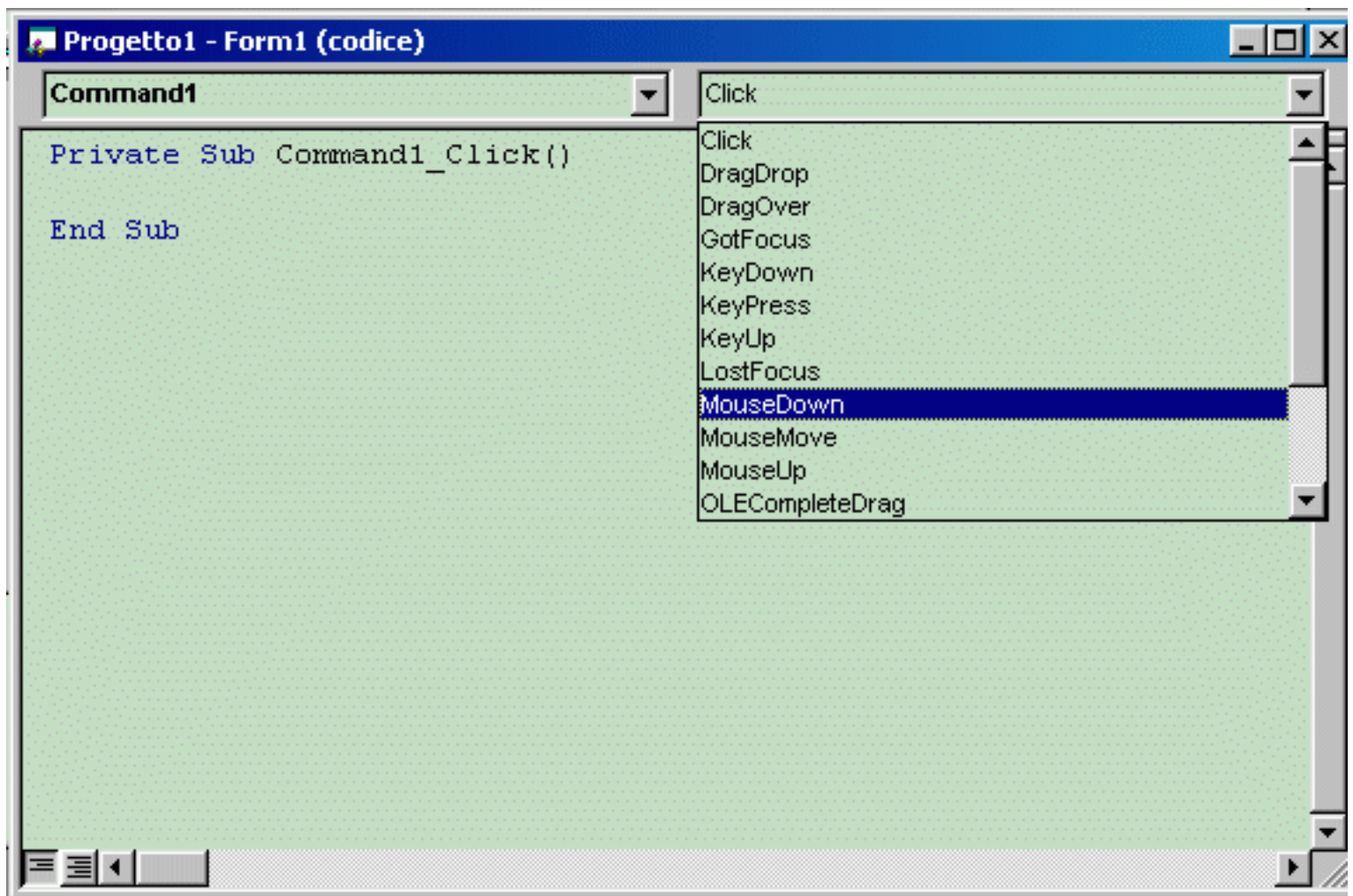
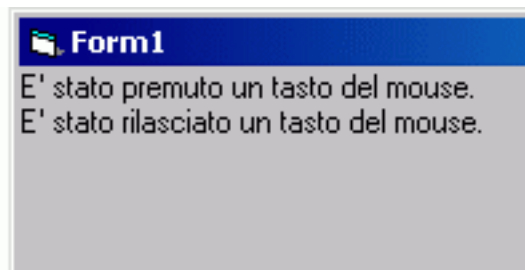
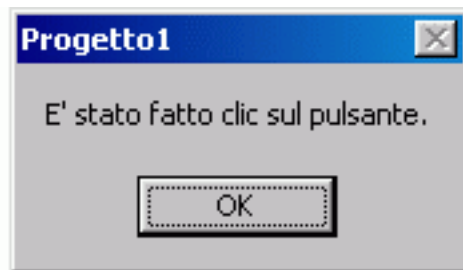
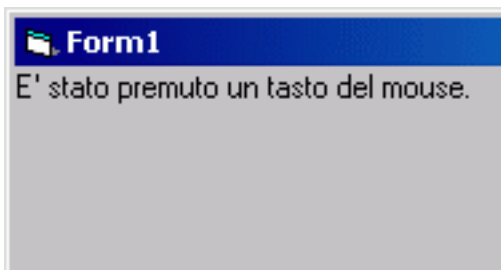
sul menu **Esegui**, quindi sul comando **Avvia**; premete il pulsante **Avvia** nella barra degli strumenti. Apparirà il form con al centro il pulsante che abbiamo inserito; fate clic su di esso: se non avete commesso errori, verrà visualizzata una finestra.

Visualizza la finestra

A questo punto, per chiudere il form, fate clic sulla **X**, a destra nella barra del titolo (come in una normale applicazione per Windows). L'esempio che abbiamo appena realizzato è disponibile per il download facendo clic qui.

Nella maggior parte dei casi, gli eventi *Click e DblClick* sono più che sufficienti per la gestione del mouse, ma in alcuni casi potrà essere necessario sapere quando si preme un pulsante del mouse, quando lo si rilascia oppure quando si sposta il mouse su un controllo: questi eventi sono chiamati, rispettivamente, *MouseDown, MouseUp e MouseMove*. Per spostarsi negli eventi *MouseDown e MouseUp*, fate doppio clic sul pulsante: verrà visualizzato il codice che abbiamo scritto precedentemente. Ora fate clic sulla lista di sinistra per aprire l'elenco e selezionate, ad esempio, *MouseDown*: sarà ora possibile scrivere il codice che si vuole venga eseguito quando si verifica questo evento. Riprendiamo dunque l'esempio precedente e aggiungiamo alcune istruzioni che ci dicano quando un pulsante del mouse viene premuto e quando, invece, rilasciato:

```
Private Sub Command1_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
Me.Print "E' stato premuto un tasto del mouse."
End Sub
Private Sub Command1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
Me.Print "E' stato rilasciato un tasto del mouse." End Sub
```

In questo esempio ci sono due cosa da spiegare. La prima è la parola chiave Me, che indica il form corrente, cioè quello in cui si sta eseguendo l'operazione (approndiremo questo concetto nella lezione 8). Dopo il punto, viene utilizzato il metodo **Print**, che in questo caso ha lo scopo di stampare del testo direttamente sopra il form. Ora eseguite il programma; in tal modo, oltre a verificare in prima persona come funzionano questi eventi, vedrete anche l'ordine in cui essi vengono generati: prima l'evento MouseDown, poi Click e infine MouseUp. Prima vedrete sul form la scritta *E' stato premuto un tasto del mouse*, subito dopo comparirà la finestra di messaggio *E' stato fatto clic sul pulsante* e, infine, di nuovo sul form, *E' stato rilasciato un tasto del mouse*:

Visualizza le finestre

Se volete scaricare il nuovo esempio, fate clic qui. Nella prossima lezione ci occuperemo dei principali eventi che vengono generati dalla tastiera, per poi passare finalmente ad analizzare con più attenzione i controlli di VB, a cominciare dal form.

Lezione successiva
[Sommario]
TORNA SU

Progetto1 - Form1 (codice)
Command1
Click
Private Sub Command1_Click()
End Sub
Click
DragDrop
DragOver
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus
MouseDown
MouseMove
MouseUp
OLECompleteDrag

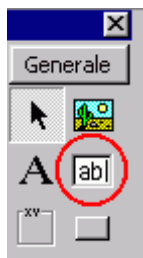
Form1
E' stato premuto un tasto del mouse.

Progetto1
E' stato fatto clic sul pulsante.
OK

Form1
E' stato premuto un tasto del mouse.
E' stato rilasciato un tasto del mouse.

GUIDA AL VISUAL BASIC

LEZIONE 7: *Gli eventi fondamentali della tastiera*

Gli eventi fondamentali della **tastiera** sono 4: Change (che però, come vedremo più avanti, può essere generato anche con un'azione del mouse), *KeyPress, KeyDown e KeyUp*. L'evento *Change* viene generato quando si modifica il contenuto di un controllo; nel controllo *TextBox* (nell'immagine a lato è evidenziata la sua posizione nella Casella degli strumenti), ad esempio, tale evento viene scatenato quando cambia il testo in esso contenuto. Si deve fare attenzione all'uso di questo evento in quanto può innescare degli eventi a catena difficile da prevedere che posso mandare in loop l'applicazione. Ad esempio, se all'interno dell'evento Change di un TextBox si scrive del codice che modifica il testo del TextBox verrà generato di nuovo l'evento Change, e così via.

Quando un utente preme un tasto viene generato l'evento *KeyDown*, poi si genera l'evento *KeyPress* e, infine, *KeyUp*. E' importante notare che l'evento KeyPress viene generato solo se si premono i tasti alfanumerici, i tasti di funzione, e i tasti ESC, Backspace e Invio; l'evento non si verifica, ad esempio, se l'utente preme le frecce direzionali.

A questo punto possiamo creare un **piccolo progetto** di esempio per verificare quanto detto finora. Per le nostre prove utilizzeremo il controllo TextBox, di cui abbiamo parlato poc'anzi.

Fate clic sull'icona che lo rappresenta nella Casella degli strumenti, poi portatevi nel punto del form in cui lo volete inserire, fate clic con il tasto sinistro del mouse e, tenendolo premuto, create la casella con la forma che desiderate. Dopo aver rilasciato il tasto del mouse, la TextBox comparirà nel form. Vogliamo ora scrivere il codice per gestire gli eventi *Change, KeyPress, KeyUp e KeyDown*. Per fare questo, innanzi tutto fate doppio clic sul controllo: comparirà la solita finestra in cui scrivere il codice del programma L'evento in cui ci troviamo è proprio Change:

Private Sub Text1_Change()
End Sub

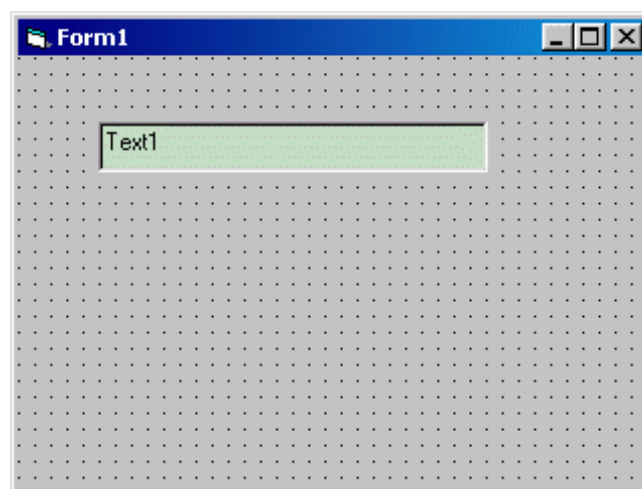
All'interno di questa routine scrivete semplicemente

Me.Print "Il contenuto della casella di testo è cambiato."

Il significato della parola Me e del metodo Print sono già stati accennati nella

lezione precedente; la parola chiave Me sarà discussa più approfonditamente nella lezione successiva. Ora premete F5 per avviare il programma e provate a digitare qualcosa nella casella di testo: noterete che, ogni volta che viene premuto un tasto, sul form compare la scritta *Il contenuto della casella di testo è cambiato*. Dopo aver chiuso il programma con un clic sulla **X** nella barra del titolo, completiamo l'esempio inserendo questo codice:

```
Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer) Me.Print
"Evento KeyDown"
End Sub
Private Sub Text1_KeyPress(KeyAscii As Integer)
Me.Print "Evento KeyPress"
End Sub
Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)
Me.Print "Evento KeyUp"
End Sub
```

Eseguite nuovamente il programma e, come prima, digitate qualcosa nella casella di testo; osservate quello che viene scritto sul form di volta in volta. Potete prelevare l'esempio che abbiamo appena realizzato facendo clic qui.

Abbiamo così analizzato gli eventi fondamentali generati dalla tastiera. Ora possiamo iniziare a parlare più dettagliatamente dell'elemento principale di un'applicazione: il form, l'oggetto della prossima lezione.

HTML.IT SHOP| CORSI IN AULA| FREE INTERNET| WEBTOOL BLOG| CREA| DOWNLOAD| FORUM| LIBRI| NEWSLETTER ADSL| VOIP| HOSTING ASP| B2B| FLASH-MX| FONT| GIF| LINUX| NEWS| PHP| PRO| PROGRAMMAZIONE| SICUREZZA

cerca vai

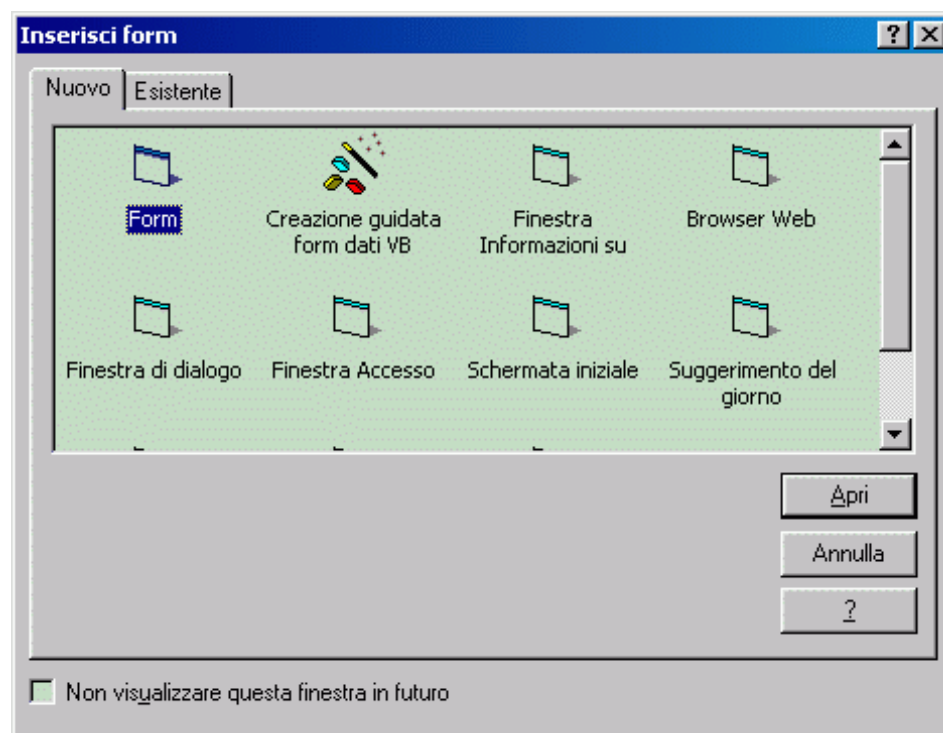
Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 8: ***Il form, l'elemento centrale di un'applicazione***

La parte fondamentale del disegno di una qualsiasi applicazione è la progettazione dell'interfaccia tra il computer e la persona che lo utilizzo. I componenti fondamentali per la creazione dell'interfaccia sono i form e i controlli.

Il **form** (letteralmente "forma", "immagine", ma in questo caso indica genericamente la "finestra" di Windows) è l'elemento centrale di ogni applicazione; un form, infatti, costituisce la base dell'interfaccia utente: è in un form che vengono inseriti tutti gli altri controlli, siano essi pulsanti, caselle di testo, timer, immagini, ecc. Un form è prima di tutto un oggetto e, come tale, dispone di proprietà che ne definiscono l'aspetto, di metodi e di eventi che permettono di determinare, tra le altre cose, l'interazione con l'utente.

La creazione di un nuovo form è diversa dalla procedura di inserimento di un controllo, che abbiamo visto nelle lezioni precedenti. Per inserire un nuovo form fate clic sul comando **Inserisci form** nel menu **Progetto** oppure fate clic sull'icona corrispondente nella barra degli strumenti (e visibile a lato). A questo punto comparirà la finestra di dialogo **Inserisci form**: per creare un form vuoto dovete fare clic sull'icona Form. In tale finestra potete inoltre notare che VB mette a disposizione diversi modelli di form predefiniti che si possono utilizzare come basi da personalizzare secondo le proprie esigenze.

Ora che sappiamo come aggiungere un nuovo form, possiamo vedere quali sono le proprietà fondamentali. Una delle più importanti è BorderStyle, che permette di definire l'aspetto del bordo della finestra. I valori che può assumere sono 5: *None, Fixed Single, Sizable, Fixed Dialog, Fixed ToolWindow e Sizable ToolWindow*. I valori più usati sono Sizable (predefinito, vengono visualizzati tutti i pulsanti, di riduzione ad icona, ingrandimento e chiusura, nonché la barra del titolo) e Fixed Dialog (form a dimensione fissa, vengono visualizzati il pulsante di chiusura e la barra del titolo). Potete provare gli altri stili della finestra, che qui sono stati solo accennati, portandovi nella finestra delle Proprietà e andando a modificare la proprietà BorderStyle. Un'altra proprietà importante è la proprietà Caption, che permette di specificare la stringa da visualizzare nella barra del titolo del form. Infine, la proprietà Icon consente di selezionare l'icona che contraddistinguerà il form. Per inserire un'icona, andate come di consueto nella finestra delle Proprietà e ciccate sul pulsante con i tre puntini che appare quando si seleziona la proprietà Icon: così facendo verrà mostrata la finestra di dialogo **Carica icona**, in cui selezionare il file dell'icona (estensione .ICO) desiderata.

Passiamo ora agli eventi più utilizzati, nell'ordine in cui vengono generati. Quando si verifica l'evento Load, tutti i controlli inseriti nel form sono stati creati, anche se la finestra non viene ancora visualizzata; è, questo, uno degli eventi più utilizzati, dal momento che di solito contiene il codice che si vuole venga eseguito quando si carica un form:

Private Sub Form_Load()

End Sub

L'evento **Resize** viene generato immediatamente prima che il form diventi visibile ed ogni volta che la finestra viene ridimensionata, sia trascinando i suoi bordi, sia ingrandendola, riducendola ad icona e ripristinandola:

Private Sub Form_Resize()

End Sub

L'evento **QueryUnload** si verifica nel momento in cui un form sta per essere scaricato, cioè rimosso dalla memoria. Tale evento può essere utilizzato per impedire che la finestra venga effettivamente chiusa, al verificarsi di particolari situazioni, impostando il parametro Cancel a True:

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
'Se imposto il parametro Cancel a True, la chiusura del form verrà annullata.
'Cancel = True
End Sub
```

Provate a inserire l'istruzione Cancel = True nella routine QueryUnload (oppure fate clic qui per prelevare l'esempio) ed eseguite l'applicazione: in questo modo, premendo il tasto di chiusura sulla barra del titolo, il form non verrà chiuso. Per terminare l'applicazione fate clic sul pulsante **Fine** nella barra degli strumenti. Ora ci restano da analizzare i metodi di cui un form dispone. Per rendere visibile un form, si deve richiamare il metodo Show:

Form1.Show

E' possibile visualizzare un form in due modalità: **Modal**, indica che il form è a scelta obbligatoria, ovvero impone all'utente di dare una risposta, premere un tasto, ecc., prima di restituire il controllo all'applicazione e, quindi, eseguire le istruzioni successive al metodo Show; Modeless: indica che il form non è a scelta obbligatoria, quindi il codice che segue il metodo **Show** viene richiamato immediatamente dopo la visualizzazione della finestra. Per visualizzare un form a scelta obbligatoria occorre specificare il parametro vbModal quando si richiama il metodo Show:

Form1.Show vbModal

Viceversa, se non specificato alcun parametro, la finestra visualizzata verrà considerata non a scelta obbligatoria (cioè Modeless).

Sviluppando un'applicazione capita praticamente sempre di dover utilizzare più form; quando vogliamo visualizzare un secondo form, ad esempio una finestra contente le opzioni del programma, dobbiamo utilizzare l'istruzione Load, che ha proprio lo scopo di caricare in memoria un form o un controllo. Ad esempio, supponiamo di avere un form principale e di volere che, alla pressione di un pulsante, venga visualizzato un altro form a scelta obbligatoria. Innanzi tutto, inseriamo un pulsante nel form; poi aggiungiamo un secondo form al progetto, come abbiamo visto all'inizio di questa lezione. Ora ci dobbiamo riportare nel primo form, fare doppio clic sul pulsante e, all'interno dell'evento Click, scrivere questo codice:

```
'Carica il form in memoria.
Load Form2
'Visualizza il form a scelta obbligatoria.
Form2.Show vbModal
```

Ora avviamo il progetto premendo il tasto F5. Verrà visualizzato il primo form; se facciamo clic sul pulsante, si aprirà la seconda finestra: notate che, se provate a fare clic sul primo form, udirete un beep, dal momento che non potete accedere ad esso prima di aver chiuso la seconda finestra. Chiudiamo il secondo form e, quindi, il primo. L'esempio è anche disponibile per il download facendo clic qui.
Da ultimo, se vogliamo, ad esempio, scaricare un form quando si preme un pulsante, dobbiamo utilizzare l'istruzione Unload:

```
Unload Form1
```

Eseguendo tale istruzione verrà generato l'evento QueryUnload, visto in precedenza, in cui è possibile annullare l'uscita. Provate ad inserire questa istruzione nell'evento Click di un pulsante, poi premete il tasto F5 per avviare il progetto e fate clic sul CommandButton: se non avete commesso errori, l'applicazione dovrebbe chiudersi. Se non avete capito qualcosa, potete sempre scaricare questo esempio. Un'ultima nota: se, all'interno di un form, si vuol fare riferimento al form stesso, invece del suo nome è possibile utilizzare la parola chiave Me: Ad esempio, invece di scrivere Unload Form1, dal momento che la finestra che vogliamo chiudere è la stessa in cui viene richiamata la funzione Unload, sarebbe stato lecito (e di solito è la pratica più comune) scrivere Unload Me.

Tramite l'impostazione delle proprietà del form, il disegno dei controlli e la scrittura di codice VB per la risposta agli eventi, è possibile creare interfacce sempre più accattivanti. Ricordate che il successo di un programma è legato sempre di più al corretto disegno dell'interfaccia.

Lezione successiva
[Sommario]
TORNA SU

HTML.IT SHOP| CORSI IN AULA| FREE INTERNET| WEBTOOL BLOG| CREA| DOWNLOAD| FORUM| LIBRI| NEWSLETTER ADSL| VOIP| HOSTING ASP| B2B| FLASH-MX| FONT| GIF| LINUX| NEWS| PHP| PRO| PROGRAMMAZIONE| SICUREZZA

cerca vai

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

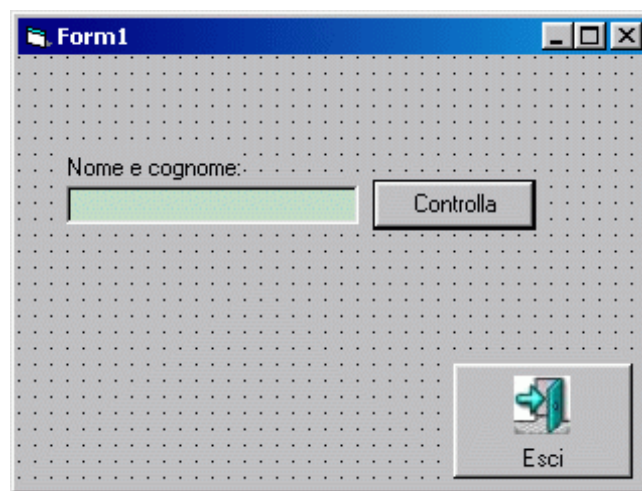
GUIDA AL VISUAL BASIC

LEZIONE 9: *I controlli di Visual Basic: CommandButton, TextBox e Label*

Comincia da questa lezione un'analisi più approfondita dei controlli messi a disposizione da Visual Basic per costruire l'interfaccia utente. Iniziamo col descrivere i controlli *CommandButton, TextBox e Label*; termineremo la lezione con un esempio che mostrerà come utilizzare questi elementi.
Del controllo CommandButton (pulsante di comando) abbiamo già parlato nelle lezioni precedenti (vedi lezione 5 e lezione 6) quando abbiamo parlato della Casella degli strumenti di VB e degli eventi generati dal mouse.

In generale, il **CommandButton** è utilizzato con una didascalia (la caption) e, opzionalmente, un'immagine che fanno comprendere immediatamente all'utente l'azione che verrà eseguita quando il pulsante sarà premuto. L'evento più utilizzato del CommandButton è il Click, ed è in esso che si dovrà scrivere il codice da eseguire alla pressione del pulsante. Per inserire un'immagine nel pulsante si deve modificare la proprietà Style in 1 - Graphical e quindi fare clic sulla proprietà Picture, in modo da far apparire la finestra di dialogo in cui selezionare l'immagine (di tipo bitmap, icona, metafile, GIF e JPG). Quest'ultima proprietà si può anche impostare in fase di esecuzione, utilizzando l'istruzione LoadPicture. Supponiamo di avere un'immagine di tipo bitmap nella cartella *C:\Icone\App.bmp;* dopo aver settato la proprietà Style su Graphical, il comando da eseguire per associare l'immagine al pulsante è:

Command1.Picture = LoadPicture("C:\Icone\App.bmp")

Impostando la proprietà *Default* su True è possibile definire un pulsante come predefinito, in modo che se l'utente preme il tasto Invio da un qualsiasi punto della finestra il controllo passi al pulsante attivando l'evento Click. In modo analogo, settando la proprietà Cancel su True, il pulsante verrà attivato quando viene premuto il tasto Esc (questo è il tipico caso di un pulsante Annulla).
Il controllo **TextBox** (casella di testo) offre all'utente la possibilità di visualizzare, modificare e immettere informazioni. Quando si inserisce un TextBox in un form, in esso viene visualizzato il nome del controllo; per modificarne il contenuto è sufficiente utilizzare la proprietà Text, disponibile nella finestra delle Proprietà e accessibile da codice:

Text1.Text = "Testo di prova"

Per modificare le modalità di visualizzazione si possono usare le proprietà Alignment e Font. La prima consente di impostare l'allineamento del testo all'interno della casella è può assumere i seguenti valori: 0 - Left Justify (testo allineato a sinistra); 1 - Right Center (allineato a destra); 2 - Center (centrato). La proprietà Font, invece, è usata per modificare il carattere con cui viene visualizzato il testo: Facendo clic su tale proprietà viene visualizzata una finestra di dialogo per selezionare il tipo di carattere tra quelli installati nel computer, la dimensione, lo stile e gli effetti. Se si desidera modificare il colore del testo è necessario modificare la proprietà ForeColor. Per impostazione predefinita una casella di testo non permette di andare a capo, ma il testo viene visualizzato su un'unica riga che scorre verso destra nel momento in cui si raggiunge il margine del controllo. Per consentire la digitazione su più righe è sufficiente impostare la proprietà MultiLine su True e la proprietà ScrollBars su 3 - Both, così da mostrare le barre di scorrimento che consentono di visualizzare un testo più lungo rispetto alle dimensioni del controllo. Infine, se si vuole limitare la lunghezza massima del testo che può essere digitato in una TextBox basta impostare la proprietà MaxLength sul numero massimo di caratteri che si possono immettere.

Il controllo **Label** (etichetta) è solitamente usato per rendere più esplicativa l'interfaccia, ad esempio come didascalia o commento di una casella di testo; a differenza di quest'ultima, l'utente non può modificare direttamente il testo visualizzato in una Label. Un'altra diversità è che per impostare il testo di un'etichetta si deve usare la proprietà Caption e non Text, che non è presente. Un aspetto comune tra i due controlli, invece, è che anche in una Label, appena inserita in un form, viene visualizzato il nome del controllo. E' possibile fare in modo che il controllo si ridimensioni automaticamente per adattarsi alla lunghezza del testo impostando la proprietà Autosize su True; la proprietà WordWrap, infine, permette l'estensione del testo su più righe.

Realizziamo ora un piccolo esempio per mettere in pratica quanto abbiamo detto finora. Vogliamo creare una piccola applicazione con una casella di testo in cui si deve immettere il proprio nome e un pulsante che controlla la validità del testo immesso. Per rendere l'interfaccia più amichevole aggiungiamo anche un'etichetta per informare l'utente sul tipo di dati che deve immettere e un pulsante con un'icona per uscire dal programma.

Innanzi tutto inseriamo un controllo **TextBox** nel form; al suo interno verrà visualizzato il nome del controllo, in questo caso Text1. Eliminiamo questo testo modificando la proprietà Text. Ora a sinistra della casella di testo posizioniamo un pulsante e modifichiamo la sua proprietà Caption su Controlla; modifichiamo inoltre la sua proprietà Default settandola su True. Ora aggiungiamo l'etichetta esplicativa: creiamo una Label sopra la casella di testo e impostiamo la sua Caption su Nome e cognome:. Infine aggiungiamo un pulsante nell'angolo in basso a destra del form, modifichiamo il sua Caption su Esci, la proprietà Cancel su True e Style su 1 - Graphical. Ora fate clic sulla proprietà *Picture del CommandButton* e selezionate l'icona da visualizzare sul pulsante (se non avete un'icona adatta, la potete scaricare facendo clic qui).
Dopo queste operazioni il form dovrebbe risultare come quello mostrato a lato. Ora dobbiamo scrivere il codice del programma. Vogliamo che, premendo il

tasto Controlla, venga controllata la proprietà Text della casella di testo e venga visualizzato un messaggio diverso a seconda che in essa sia contenuto o no del testo. Infine, premendo il tasto Esci, l'applicazione deve terminare. Nella routine Command1_Click andremo a scrivere:

```
'Controlla la proprietà Text.
If Text1.Text = "" Then MsgBox "Digitare il nome e il cognome." Else MsgBox "Valori corretti."
```

Questa semplice istruzione controlla il valore della proprietà Text; se è uguale a "" (stringa vuota), significa che non è stato digitato nulla, quindi visualizza un messaggio che chiede di inserire i dati richiesti; altrimenti, qualora sia stato digitato qualcosa, informa l'utente che i valori sono corretti. L'ultima cosa che ci resta da fare è scrivere il codice per chiudere il programma quando si preme il tasto **Esci**. Come abbiamo visto nella lezione precedente, per fare questo basta scrivere nell'evento Command2_Click l'istruzione Unload Me.
Adesso complichiamo il programma. Vogliamo fare in modo che, se l'utente preme il tasto **Esci** senza aver digitato nulla nella casella di testo, venga visualizzata una finestra di dialogo e l'uscita venga annullata. Possiamo raggiungere questo obiettivo scrivendo il codice di controllo nell'evento Form_QueryUnload e, se necessario, impostando il parametro Cancel su True per annullare l'uscita:

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
'Controlla se nella casella è stato digitato qualcosa.
If Text1.Text = "" Then
'Non è stato digitato nulla; visualizza un messaggio.
MsgBox "Immettere il nome e il cognome per uscire."
'Sposta lo stato attivo sul controllo TextBox.
Text1.SetFocus
'Annulla l'uscita.
Cancel = True
End If
End Sub
```

L'unica istruzione che merita qualche commento è **Text1.SetFocus**. Il metodo SetFocus ha lo scopo di spostare lo stato attivo sul controllo, cioè, nel caso specifico, di visualizzare il cursore all'interno della TextBox, cosicché sia possibile digitare al suo interno. L'esempio completo che abbiamo realizzato in questa lezione è disponibile per il download facendo clic qui.

Lezione successiva

[Sommario]

TORNA SU

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 10: *I controlli Frame, CheckBox e OptionButton*

Continuiamo il nostro esame dei controlli standard di Visual Basic: in questa lezione ci occupiamo dei controlli *Frame, CheckBox e OptionButton*.

Il controllo Frame (cornice) permette di raggruppare elementi dell'interfaccia logicamente legati fra loro; in un Frame, ad esempio, possiamo inserire tutte le opzioni relative al salvataggio dei file, oppure alla personalizzazione dell'interfaccia, ecc. La comodità del Frame è che tutti gli elementi inseriti in esso vengono trattati come un "blocco" unico; ad esempio, se si nasconde un Frame, verranno automaticamente nascosti anche tutti i controlli al suo interno. Per inserire un elemento in un Frame è necessario disegnarlo all'interno della cornice stessa; fatto questo, sarà possibile spostarlo solo entro i margini del Frame. Trattandosi di un contenitore di altri oggetti, di solito non vengono gestiti i suoi eventi, ma ci si limita a modificare poche proprietà, innanzi tutto la Caption, per modificare l'etichetta del controllo.

Il controllo CheckBox (casella di controllo) è rappresentato graficamente da un'etichetta con a fianco una casella di spunta, nella quale viene visualizzata una crocetta quando viene selezionato. Solitamente è utilizzato per visualizzare una serie di opzioni tra cui selezionare quelle desiderate. Anche il CheckBox dispone della proprietà Caption, per modificarne l'etichetta. La proprietà più importante di questo controllo è la Value, che permette di impostare o recuperare lo stato del CheckBox, cioè di sapere se esso è selezionato oppure no. I valori che può assumere sono tre: 0 - Unchecked, il controllo non è selezionato; 1 - Checked, il controllo è selezionato; 2 - Grayed, il controllo è disabilitato nello stato di selezionato. Per cambiare lo stato del controllo è possibile modificare la proprietà Value anche da codice:

```
Check1.Value = vbUnchecked 'Deseleziona il controllo.
Check1.Value = vbChecked 'Seleziona il controllo.
Check1.Value = vbGrayed 'Deseleziona il controllo nello stato di selezionato.
```

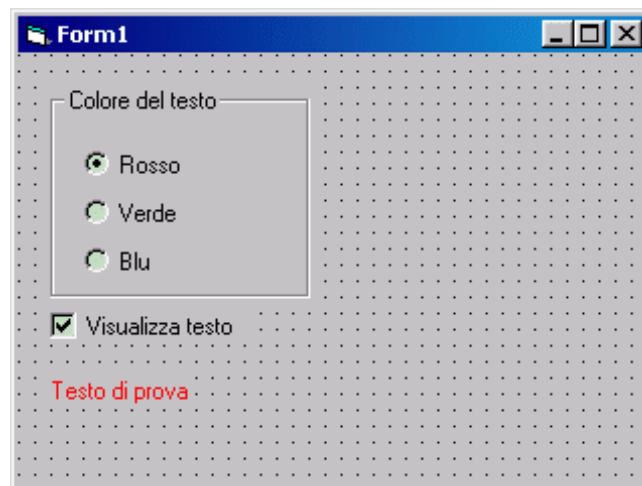
vbUnchecked, vbChecked, vbGrayed sono costanti definite da Visual Basic e valgono rispettivamente 0, 1 e 2; si possono quindi usare senza differenza le costanti, come mostrato sopra, oppure i loro valori numerici. L'evento più utilizzato del CheckBox è l'evento Click, che si verifica quando si modifica lo stato del controllo; di solito, in tale evento si inserisce il codice per attivare o disattivare altri controlli in accordo con la scelta effettuata. Come esempio, diamo un'occhiata a questa routine:

```
Private Sub Check1_Click()
```

```
If Check1.Value = vbChecked Then
'Il controllo è stato selezionato.
Check1.Caption = "Il controllo è stato selezionato."
Else
'Il controllo è stato deselezionato.
Check1.Caption = "Il controllo è stato deselezionato."
End If
End Sub
```

Quando si verifica l'evento Click, viene controllato se il controllo è stato selezionato o deselezionato e, quindi, cambia in modo opportuno la Caption del CheckBox. E' importante notare che l'evento Click viene generato anche quando si modifica la proprietà Value da codice.

Il controllo OptionButton (pulsante di opzione) è simile al CheckBox, ma con la differenza che in un gruppo di controlli OptionButton è possibile selezionare un solo elemento alla volta, mentre è possibile selezionare più controlli CheckBox contemporaneamente. Anche per questo tipo di controlli le proprietà più utilizzate sono la Caption e la Value, che in tal caso assume i valori True (controllo selezionato o False (controllo non selezionato). Solitamente gli OptionButton sono raggruppati in un altro controllo, come un Frame, perché VB presume che tutti i pulsanti di opzione presenti nello stesso form appartengano al medesimo gruppo; di conseguenza è possibile selezionare solo un OptionButton alla volta tra quelli presenti in uno stesso gruppo. Anche l'OptionButton dispone dell'evento Click, che si verifica quando si modifica il suo stato, ovvero quando viene selezionato facendo clic su di esso.

Come di consueto vediamo ora con un esempio di mettere in pratica quanto abbiamo detto in questa lezione. Vogliamo creare un semplice programma che visualizzi, all'interno di un Frame, una serie di opzioni per modificare il colore del testo di una Label. Una casella di controllo, poi, deve permettere di visualizzare o nascondere l'etichetta. Per semplicità, d'ora in poi indicheremo solo gli elementi che si vogliono inserire nel form e quali proprietà andranno modificate; è possibile fare riferimento all'immagine visualizzata a lato per avere un'idea della disposizione dei controlli. E' possibile, inoltre, scaricare un form che contiene solo gli elementi dell'interfaccia utente, nel caso non si riesca a ricreare la disposizione dei controlli proposta.

Ora dobbiamo inserire il codice del nostro programma. Intuitivamente, vogliamo che quando l'utente esegue un clic su uno degli OptionButton, il colore del testo venga modificato. Per intercettare la pressione del tasto del mouse sopra il controllo sopra menzionato dobbiamo utilizzare l'evento Click; per modificare il colore del testo, invece, è necessario modificare la proprietà ForeColor dell'etichetta Label1. Detto questo, il codice diventa quasi autoesplicativo:

```
Private Sub Option1_Click()
'Si è scelto di visualizzare il testo in rosso.
Label1.ForeColor = vbRed
End Sub
```

```
Private Sub Option2_Click()
'Si è scelto di visualizzare il testo in verde.
Label1.ForeColor = vbGreen
End Sub

Private Sub Option3_Click()
'Si è scelto di visualizzare il testo in blu.
Label1.ForeColor = vbBlue
End Sub
```

Anche in questo caso sono state utilizzate le costanti definite da Visual Basic per i codici dei colori; nella Guida in linea è possibile trovare tutte le informazioni a riguardo. Ora però vogliamo completare l'esempio aggiungendo il codice che vogliamo venga eseguito quando si fa clic sul controllo Check1. In questo caso, sempre nell'evento Click, dobbiamo verificare lo stato del controllo (cioè se è selezionato oppure no) e, di conseguenza, visualizzare o nascondere la Caption:

```
Private Sub Check1_Click()
If Check1.Value = vbChecked Then
'La casella di controllo viene selezionata; visualizza la Caption.
Label1.Visible = True
Else
'La casella di controllo viene deselezionata; nasconde la Caption.
Label1.Visible = False
End If
End Sub
```

Ora non ci resta che premere il tasto F5 per avviare il progetto, che è anche disponibile per il download facendo clic qui.

Lezione successiva

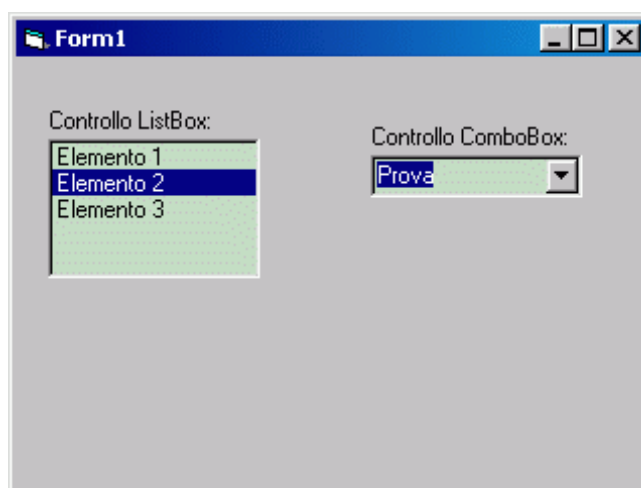
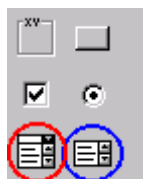
[Sommario]

TORNA SU

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 11: *I controlli ListBox e ComboBox*

In questa lezione ci occuperemo di altri due controlli di VB molto usati: *ListBox, e ComboBox*. Il **ListBox** (cerchiato in blu) e il **ComboBox** (cerchiato in rosso) sono utilizzati con lo stesso scopo, cioè visualizzare una lista di opzioni selezionabili dall'utente; la differenza fondamentale tra i due controlli è il modo in cui la lista è presentata. Nel ListBox, infatti, gli elementi della lista sono sempre visibili, mentre nel ComboBox la lista è "a scomparsa", ovvero è visibile soltanto se l'utente fa clic sulla freccia verso il basso a destra del controllo; nel ComboBox, inoltre, è possibile digitare un valore che non compare nella lista di quelli disponibili.

Per il resto, i due controlli si comportano nello stesso modo e molte proprietà e metodi funzionano allo stesso modo. Per aggiungere valori ad un ListBox o ad un ComboBox si possono seguire due strade: utilizzare la proprietà **List** disponibile nella Finestra delle proprietà oppure il metodo **AddItem** richiamabile da codice. Vediamo un esempio di questa seconda soluzione:

```
List1.AddItem "Marco"
List1.AddItem "Luigi"
List1.AddItem "Andrea"
```

Tali istruzioni aggiungono tre valori in una ListBox; lo stesso identico codice funziona con una ComboBox, basta cambiare il nome dell'oggetto. Per eliminare tutte le voci da un elenco si deve richiamare il metodo **Clear**:

List1.Clear

Se, invece, si desidera eliminare un ben preciso elemento, è necessario conoscerne la posizione all'interno della lista (il cosiddetto indice). A questo proposito, è importante ricordare che l'indice delle voci ha base 0, ovvero il primo elemento di una lista ha indice 0, il secondo ha indice 1, il terzo 2, e così via. Se, ad esempio, vogliamo rimuovere il sesto elemento contenuto in un controllo ListBox, l'istruzione da utilizzare è:

List1.RemoveItem 5

Se l'indice specificato non esiste (ad esempio si tenta di cancellare il quinto elemento di una lista che ha solo quattro elementi) verrà generato un errore. Per recuperare l'indice della voce selezionata dall'utente si deve controllare la proprietà ListIndex, sia per le ListBox sia per le ComboBox; anche in questo caso il valore restituito parte da 0. Se nessun elemento della lista è stato selezionato, il valore restituito da ListIndex sarà -1. Questa proprietà può anche essere utilizzata per selezionare da codice un particolare elemento della lista; ad esempio, se vogliamo selezionare il secondo elemento di una ListBox, è sufficiente scrivere:

List1.ListIndex = 1

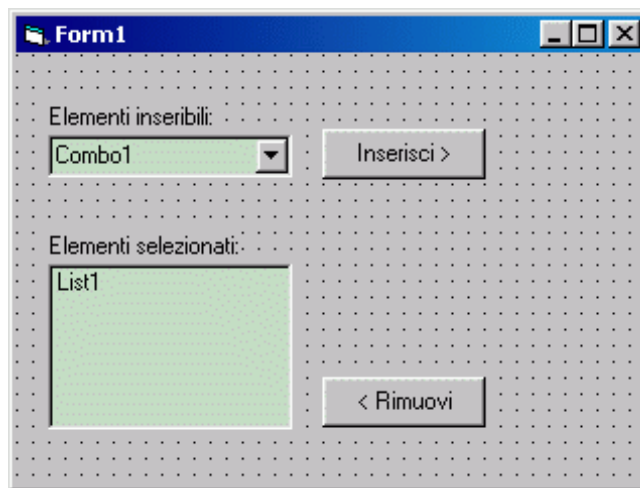
Detto questo, è molto semplice rimuovere l'elemento selezionato in una lista; per compiere tale operazione, infatti, basta il comando:

List1.RemoveItem List1.ListIndex

La proprietà Text restituisce la stringa selezionata in un ListBox o visualizzata in un ComboBox. Se, invece, vogliamo conoscere il valore di un particolare elemento di una lista, possiamo utilizzare la proprietà List, che richiede come parametro l'indice dell'elemento che si vuole recuperare. Ad esempio:

MsgBox List1.List(2)

Visualizza una finestra di messaggio contenente il valore del terzo elemento della lista.

Come abbiamo già detto, le proprietà e i metodi sopra esposti sono comuni sia al controllo ListBox sia al controllo ComboBox. Qualche parola va però ancora spesa a proposito della proprietà Style del ComboBox. Essa può assumere tre valori: 0 - Dropdown combo (casella combinata a discesa, predefinita), comprende una casella di riepilogo a discesa e una casella di testo; l'utente potrà eseguire una selezione nella casella di riepilogo o digitare nella casella di testo; 1 - Simple combo (casella combinata semplice), comprende una casella di testo ed una casella di riepilogo non a discesa, cioè sempre visibile; l'utente potrà eseguire una selezione nella casella di riepilogo o digitare nella casella di testo. Le dimensioni di una casella combinata semplice si riferiscono a entrambi i componenti. Per impostazione predefinita, la casella è dimensionata in modo che nessun elemento dell'elenco sia visualizzato; per visualizzare elementi dell'elenco, aumentare il valore assegnato alla proprietà Height; 2 - Dropdown list (casella di riepilogo a discesa), questo stile consente soltanto la selezione dall'elenco a discesa.

Ora che abbiamo visto come funzionano i controlli **ListBox e ComboBox**, facciamo un esempio pratico del loro utilizzo, così da fissare meglio i concetti che sono stati trattati. La nostra applicazione sarà così strutturata: una ComboBox conterrà un elenco di elementi; selezionandone uno e facendo clic su un pulsante, la voce selezionata verrà inserita in una ListBox e rimossa dal ComboBox; un altro pulsante, poi, permetterà di rimuovere l'elemento dalla ListBox e di reinserirlo nella ComboBox. Come sempre, potete scaricare il form contenente solamente gli elementi dell'interfaccia nella giusta posizione, senza il codice.

Prima di tutto dobbiamo popolare il ComboBox con gli elementi che possono essere selezionati dall'utente; utilizzeremo il metodo *AddItem* nell'evento *Form_Load*, che, come abbiamo già detto, viene eseguito quando il form viene caricato in memoria. Inseriamo quindi un certo numero di voci:

```
Private Sub Form_Load()
Combo1.AddItem "Scheda madre"
Combo1.AddItem "Processore"
Combo1.AddItem "Monitor"
Combo1.AddItem "Videocamera"
Combo1.AddItem "Stampante"
Combo1.AddItem "Scanner"
'Seleziona il primo elemento.
Combo1.ListIndex = 0
End Sub
```

Ora vogliamo fare in modo che, premendo il pulsante **Inserisci**, l'elemento corrente venga inserito nel ListBox e, subito dopo, venga rimosso dal ComboBox:

```
Private Sub Command1_Click()
'Inserisce l'elemento selezionato nel ListBox.
List1.AddItem Combo1.Text
'Rimuove l'elemento dal ComboBox.
Combo1.RemoveItem Combo1.ListIndex
If Combo1.ListCount > 0 Then
'Se ci sono ancora elementi, seleziona il primo elemento del ComboBox.
Combo1.ListIndex = 0
Else
'Altrimenti, disattiva il pulsante "Inserisci".
Command1.Enabled = False
End If
End Sub
```

Ogni istruzione è commentata, è necessario spendere qualche parola solo sul ciclo If… Then… Else. Dopo l'inserimento di una voce, l'elemento viene rimosso dal ComboBox; a questo punto, la routine controlla quanti elementi sono rimasti, utilizzando la proprietà ListCount: se è maggiore di 0, cioè se sono presenti ancora voci nella lista, seleziona la prima, altrimenti disattiva il pulsante, poiché non possono essere selezionati e quindi inseriti altri elementi.

Ora dobbiamo inserire il codice che verrà eseguito premendo il tasto **Rimuovi**: facendo clic su tale pulsante l'elemento selezionato nella ListBox verrà rimosso e reinserito nella ComboBox:

```
Private Sub Command2_Click()
'Innanzi tutto, controlla se è stato selezionato un elemento nel ListBox.
If List1.ListIndex >= 0 Then
'Reinserisce la voce nel ComboBox.
Combo1.AddItem List1.Text
'Rimuove l'elemento dal ListBox.
List1.RemoveItem List1.ListIndex
'Riattiva il pulsante "Inserisci".
Command1.Enabled = True
Combo1.ListIndex = 0
End If
End Sub
```

Notiamo che il codice viene eseguito solo se la proprietà ListIndex del ListBox è maggiore o uguale a 0, ovvero se è stato selezionato un elemento nella lista. Abbiamo così scritto tutto il codice necessario al funzionamento del nostro esempio, che come sempre potete scaricare facendo clic qui .

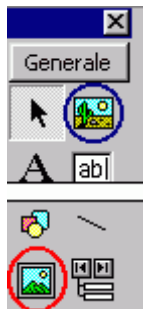
Lezione successiva

[Sommario]

TORNA SU

HTML.it
PROGRAMMAZIONE

HTML.IT SHOP| CORSI IN AULA| FREE INTERNET| WEBTOOL BLOG| CREA| DOWNLOAD| FORUM| LIBRI| NEWSLETTER ADSL| VOIP| HOSTING ASP| B2B| FLASH-MX| FONT| GIF| LINUX| NEWS| PHP| PRO| PROGRAMMAZIONE| SICUREZZA

cerca vai

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 12: ***I controlli ImageBox e PictureBox***

I controlli *ImageBox* (immagine - cerchiato in rosso) e *PictureBox* (casella immagine - cerchiato in blu) consentono di visualizzare immagini nella propria applicazione. Questi due oggetti sono per alcuni versi simili e dispongono di proprietà analoghe. Il controllo PictureBox, però dispone di numerose proprietà in più e può anche essere utilizzato come contenitore di altri controlli, alla stregua di un controllo Frame, di cui abbiamo parlato nella lezione 10: allo scopo, anche in questo caso sarà sufficiente creare un oggetto all'interno di una PictureBox. Per visualizzare un'immagine nel controllo PictureBox si usa la proprietà **Picture**, disponibile nella finestra delle proprietà del controllo: basta fare clic sui tre puntini a destra del campo *Picture* per visualizzare la finestra **Carica immagine**, in cui è possibile selezionare il file da caricare; i formati supportati sono BMP, GIF, JPG, WMF, EMF, ICO e CUR.

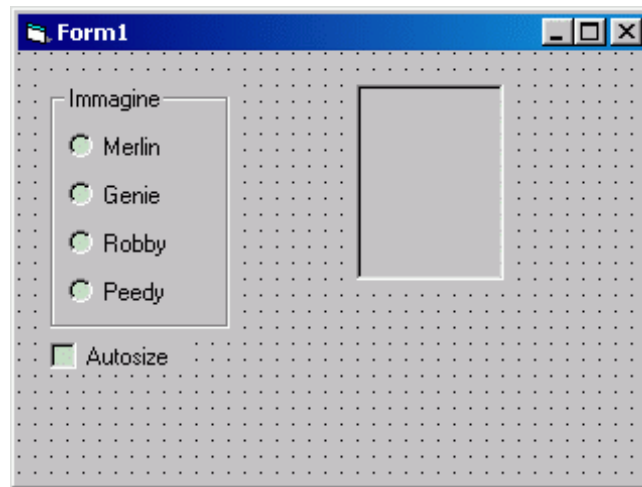
Per caricare un'immagine da codice, invece, si usa la funzione LoadPicture, come nel seguente esempio:

Picture1.Picture = LoadPicture("C:\Documenti\Prova.bmp")

Questa istruzione carica il file **Prova.bmp**, contenuto nella cartella *C:\Documenti*, e lo visualizza nel controllo Picture1. Una proprietà importante di questo controllo è AutoSize, che consente di impostare il tipo di ridimensionamento da adottare: se AutoSize è True, il controllo si ridimensiona automaticamente per adattarsi alle dimensioni dell'immagine caricata.

Come già accennato, il controllo **Image** è più semplice del PictureBox, infatti supporta solo alcune proprietà eventi e metodi del PictureBox e non può essere utilizzato come contenitore di altri controlli. Per visualizzare un'immagine, si possono usare la proprietà Picture o la funzione LoadPicture, in modo analogo al PictureBox. Non esiste la proprietà **AutoSize**, al cui posto si può usare la proprietà **Stretch**: se è False il controllo Image si ridimensiona adattandosi all'immagine caricata; se, invece, è True, l'immagine assume le dimensioni del controllo, quindi potrebbe risultare distorta.

Come potete notare, il principio di funzionamento di questi due controlli è semplice. Cerchiamo di mettere in pratica quanto fin qui detto. Creiamo un progetto composto da un frame, al cui interno di trovano 4 OptionButton, e una

CheckBox e un controllo Picture: vogliamo che, a seconda del pulsante di opzione selezionato, venga visualizzata una diversa immagine; la casella di controllo, poi, consente di adattare le dimensioni della PictureBox a quelle dell'immagine. Se volete utilizzare le stesse immagini di questo esempio per le vostre prove, le potete scaricare facendo clic qui; tali immagini vanno poi inserite nella medesima cartella del progetto Visual Basic che andremo a creare. Allo stesso modo, potete scaricare il form contenente solamente gli elementi dell'interfaccia nella giusta posizione, senza il codice.

Nella Lezione 10 abbiamo già realizzato una applicazione che utilizzava degli *OptionButton* per consentire all'utente di compiere determinate scelte; in questo caso vogliamo che, facendo clic su ciascuno di essi, venga visualizzata una diversa immagine nel controllo PictureBox. Ecco il codice che realizza quanto detto:

```
Private Sub Option1_Click()
Picture1.Picture = LoadPicture(App.Path & "\Merlin.gif")
End Sub
Private Sub Option2_Click()
Picture1.Picture = LoadPicture(App.Path & "\Genie.gif")
End Sub
Private Sub Option3_Click()
Picture1.Picture = LoadPicture(App.Path & "\Robby.gif")
End Sub
Private Sub Option4_Click()
Picture1.Picture = LoadPicture(App.Path & "\Peedy.gif")
End Sub
```

Notate che in questo codice è stata usata la proprietà App.Path, che restituisce il percorso completo in cui è memorizzato il progetto VB; ad esempio, se il file VBP del progetto è salvato nella cartella C:\Documenti \Lavori, la proprietà *App.Path* restituirà proprio C:\Documenti\Lavori. E' stato inoltre utilizzato l'operatore &, che ha lo scopo di concatenare, cioè unire, due stringhe distinte. Ora l'unica cosa che resta da fare è scrivere il codice per attivare la proprietà AutoSize del controllo PictureBox; questo codice andrà inserito nell'evento Click del CheckBox:

```
Private Sub Check1_Click()
If Check1.Value = vbChecked Then Picture1.AutoSize = True
End Sub
```

La scrittura del codice è terminata. Premete F5 per avviare il programma: se non avete commesso errori, facendo clic su uno qualunque dei pulsanti di opzione verrà visualizzata un'immagine diversa; facendo clic su Autosize, infine, il controllo PictureBox verrà ridimensionato per adattarsi alle dimensioni dell'immagine. Fate clic qui per scaricare l'esempio completo (senza le immagini).

GUIDA AL VISUAL BASIC

LEZIONE 13: ***I controlli DriveListBox, DirListBox e FileListBox***

I controlli *DriveListBox* (cerchiato in rosso), *DirListBox* (blu), *FileListBox* (verde), e consentono di accedere ai dischi installati nel sistema e alle informazioni in essi contenute. Di solito vengono utilizzati insieme, allo scopo di fornire un sistema di navigazione tra le risorse del sistema; grazie alle proprietà di cui dispongono, infatti, è semplice sincronizzare questi tre controlli.

Il controllo DriveListBox (casella di riepilogo dei drive) visualizza le unità di memorizzazione presenti nel sistema (floppy, dischi rigidi, lettori di CD-ROM, ecc.). Per cambiare l'unità selezionata si usa la proprietà Drive, che può essere modificata solo da codice (cioè disponibile solo in fase di esecuzione):

Drive1.Drive = "D:"

Specificando una lettera di unità non valida o non disponibile, verrà generato un errore. Quando si modifica l'unità selezionata nel controllo, viene generato l'evento **Change**.

Il controllo DirListBox (casella di riepilogo delle directory) visualizza le cartelle presenti nell'unità selezionata; facendo doppio clic sul nome di una directory è possibile spostarsi al suo interno. La cartella corrente può essere modificata utilizzando la proprietà Path, che come la proprietà Drive del DriveListBox è disponibile solo in fase di esecuzione:

Dir1.Path = "C:\Documenti"

Visualizza la schermata

Analogamente al DriveListBox, se si specifica una cartella non valida verrà generato un errore; la modifica della directory corrente genera l'evento Change.
Infine, il controllo FileListBox (casella di riepilogo dei file) visualizza i file che si trovano nella cartella corrente. Esso dispone di alcune proprietà che consentono di selezionare quali file visualizzare. Innanzi tutto, con la proprietà Pattern è possibile stabilire i nomi dei file visualizzati; è possibile utilizzare i caratteri jolly * e ? e, inoltre, si possono specificare più criteri separandoli con un punto e virgola (senza spazi tra un criterio e l'altro). Ad esempio, win*.exe restituirà l'elenco di tutti i file eseguibili il cui nome inizia con "win", mentre *.gif;*.jpg restituirà l'elenco di tutti i file GIF e di quelli JPG. Le proprietà Archive, Hidden, Normal, ReadOnly e System consentono di stabilire se il controllo FileListBox deve visualizzare file con gli attributi, rispettivamente, di Archivio,

Nascosto, Normale, Sola lettura e File di sistema. Anche il controllo FileListBox dispone della proprietà Path, con la quale è possibile modificare la cartella di cui si vuole visualizzare il contenuto. Quando si seleziona un file viene generato l'evento Click; il nome del file corrente è conservato nella proprietà FileName del controllo FileListBox; ad esempio, se vogliamo che, quando si seleziona un file, venga visualizzata una finestra di messaggio contenente il nome del file stesso, basterà scrivere:

```
Private Sub File1_Click()
MsgBox File1.FileName
End Sub
```

Con un esempio cerchiamo ora di capire come si possono utilizzare insieme questi tre controlli; l'applicazione che vogliamo realizzare è un classico visualizzatore di immagini. Per questo avremo bisogno di un DriveListBox, un DirListBox, un FileListBox e un controllo Image; utilizzeremo anche alcune Label per rendere l'utilizzo dell'applicazione più immediato. Cominciamo innanzi tutto col costruire l'interfaccia del programma. Come di consueto, potete scaricare il form contenente solamente gli elementi dell'interfaccia nella giusta posizione, senza il codice.

Ricordando quanto detto prima, iniziamo a scrivere il codice. Selezionando una unità nel controllo **DriveListBox** viene generato l'evento Change, che dobbiamo utilizzare per aggiornare la proprietà Path del DirListBox, in modo che quest'ultimo visualizzi le cartelle dell'unità selezionata. Per raggiungere lo scopo è sufficiente scrivere:

```
Private Sub Drive1_Change()
Dir1.Path = Drive1.Drive
End Sub
```

Notate che, come abbiamo accennato all'inizio, selezionando una unità non disponibile verrà generato un messaggio di errore; tratteremo questo argomento nella lezione 15, dedicata proprio alla gestione degli errori in VB. Ora dobbiamo collegare i controlli DirListBox e FileListBox, in modo che quest'ultimo visualizzi i file della cartella selezionata; l'evento che dobbiamo utilizzare è il Change del DirListBox:

```
Private Sub Dir1_Change()
File1.Path = Dir1.Path
End Sub
```

Con queste semplici istruzioni abbiamo collegato tra loro i tre controlli; se volete verificarne il funzionamento, premete F5 e provate a navigare tra le risorse del sistema, selezionando unità e cartelle diverse. A questo punto dobbiamo ancora scrivere le istruzioni per visualizzare nell'**ImageBox** l'immagine selezionata; il codice necessario è questo:

```
Private Sub File1_Click()
Dim FileSelezionato As String

FileSelezionato = File1.Path & "\" & File1.FileName
Image1.Picture = LoadPicture(FileSelezionato)
End Sub
```

Innanzi tutto dichiariamo una nuova variabile, di tipo String, in cui memorizzeremo il nome e il percorso completo del file selezionato; per fare questo, usiamo la proprietà *Path di File1*, ad essa aggiungiamo il carattere "\" con l'operatore di concatenazione tra stringhe (&), e infine recuperiamo il nome del file con la proprietà FileName. L'ultima istruzione carica nel controllo Image1 il file dell'immagine che è stata selezionata.

L'applicazione è pronta: premete F5 e verificatene il corretto funzionamento. Per scaricare quanto abbiamo realizzato fate clic qui.

L e z i o n e   s u c c e s s i v a

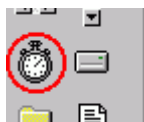
HTML.IT SHOP| CORSI IN AULA| FREE INTERNET| WEBTOOL BLOG| CREA| DOWNLOAD| FORUM| LIBRI| NEWSLETTER ADSL| VOIP| HOSTING ASP| B2B| FLASH-MX| FONT| GIF| LINUX| NEWS| PHP| PRO| PROGRAMMAZIONE| SICUREZZA

cerca vai

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 14: *Il controllo Timer*

Il controllo *Timer* consente di gestire azioni collegate al trascorrere del tempo; il suo funzionamento non dipende dall'utente e può essere programmato per l'esecuzione di operazioni a intervalli regolari. Un tipico utilizzo del Timer è il controllo dell'orologio di sistema per verificare se è il momento di eseguire una qualche attività; i timer sono inoltre molto utili per altri tipi di operazioni in background.

Il controllo *Timer* si differenzia dai controlli esaminati finora perché, in fase di esecuzione, è invisibile; dispone di un solo evento, l'evento Timer, che viene generato quando è trascorso l'intervallo di tempo (espresso in millisecondi) specificato nella proprietà *Interval*. Quest'ultima può assumere valori compresi tra 0, che equivale a disattivare il timer, e 65.535, cioè poco più di un minuto. E' buona norma impostare nella proprietà Interval un valore corrispondente a circa la metà del tempo che vogliamo trascorra effettivamente tra due eventi Timer: ad esempio, se vogliamo che l'evento Timer venga generato ogni secondo (cioè 1000 millisecondi), conviene impostare la proprietà Interval su 500, per evitare ritardi dovuti ad altre elaborazioni del computer che avvengono nello stesso momento. Un'altra proprietà importante è la proprietà Enable, che consente di stabilire o di impostare se il Timer è attivo oppure no.

Per comprende il funzionamento del controllo Timer realizzeremo un'applicazione classica, un orologio digitale. L'interfaccia, che potete scaricare facendo clic qui , è molto semplice e comprende, oltre al Timer, solo una Label.

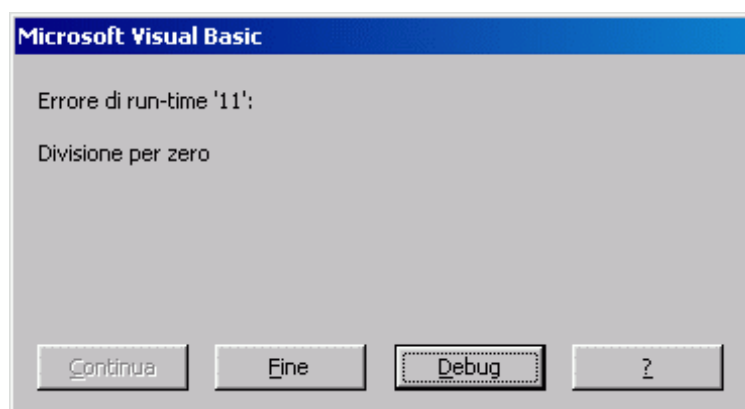
Notate che la proprietà *Interval* è stata impostata su 500 millisecondi per la ragione sopra indicata. L'unica riga di codice che dobbiamo scrivere va inserita nell'evento Timer:

```
Private Sub Timer1_Timer()
Label1.Caption = Time
End Sub
```

Per recuperare l'ora di sistema è stata usata la funzione **Time**. Tutto qui: il programma è finito, premete F5 per osservare il suo funzionamento. Per scaricare questa applicazione fate clic qui.

GUIDA AL VISUAL BASIC

LEZIONE 15: ***La gestione degli errori in Visual Basic***

Quando si esegue un programma, anche il più semplice, c'è sempre la possibilità che accada qualcosa che non è stato previsto durante la progettazione, cioè che si verifichi un errore, una situazione inaspettata che l'applicazione non è in grado di gestire. Quando si presenta un problema del genere, il programma visualizza un messaggio e, subito dopo, termina la sua esecuzione. Un errore si verifica, ad esempio, se si cerca di effettuare una divisione per zero, se si tenta di caricare in una PictureBox un file immagine inesistente, ecc. Un programma ben sviluppato dovrebbe prevedere delle procedure in grado di risolvere gli errori più comuni; continuando gli esempi precedenti, se si effettua una divisione per zero dovrebbe apparire un messaggio che informa che l'operazione non è valida e, allo stesso modo, se ci cerca di aprire un file inesistente, l'utente dovrebbe essere avvisato e avere la possibilità di modificare la sua scelta.

In VB, la gestione degli errori si effettua utilizzando i cosiddetti gestori di errori. Il loro utilizzo è molto semplice. Per installare un gestore degli errori, è sufficiente scrivere (di solito come prima istruzione all'interno di una routine):

On Error GoTo <Etichetta>

Dove è una sorta di "segnalibro" in corrispondenza del quale inizia il codice incaricato di gestire gli errori. A questo punto se, all'interno della routine, si verifica un errore (chiamato errore di run-time), l'esecuzione passa immediatamente alla parte di codice che si trova in corrispondenza dell'etichetta definita con On Error… Provate, ad esempio, a scrivere queste istruzioni nell'evento Form_Load:

```
Dim A As Long, B As Long, C As Long
A = 5
B = 0
C = A / B 'Questa istruzione causa un errore di divisione per zero.
MsgBox "Il risultato della divisione è: " & C
```

Ora premete F5: quando VB cercherà di eseguire l'istruzione C = A / B, mostrerà una finestra di errore come quella visibile a lato, contenente due tipi di informazioni, il numero dell'errore e una breve descrizione dello stesso. Premendo il tasto Fine l'esecuzione del programma verrà terminata; facendo clic su **Debug** VB evidenzierà l'istruzione che ha causato l'errore, consentendo la sua modifica; il pulsante ?, infine, visualizza la Guida in linea relativa all'errore che si è appena verificato. Il pulsante Continua, attivo solo in certe situazioni, permette di continuare l'esecuzione del programma ignorando l'errore.

Cerchiamo di modificare il codice in modo da gestire l'errore. Come abbiamo detto, per prima cosa dobbiamo creare un gestore degli errori con l'istruzione **On Error GoTo <Etichetta>**, dopodiché dobbiamo scrivere il codice di gestione vero e proprio. Ecco come possiamo modificare l'esempio:

```
Private Sub Form_Load()
'Questa istruzione causa un errore di divisione per zero.
MsgBox "Il risultato della divisione è: " & C
Exit Sub

GestoreErrori:
'Queste istruzioni vengono eseguite quando si verifica un errore.
If Err.Number = 11 Then
'Divisione per zero.
MsgBox "Si è verificato un errore di divisione per zero. Controllare i valori
digitati e riprovare."
End If
End Sub
```

Analizziamo quanto abbiamo scritto. L'istruzione **On Error Goto GestoreErrori** crea il gestore degli errori, cioè dice al programma che, in caso di errori, deve passare ad eseguire le istruzioni scritte sotto l'etichetta GestoreErrori. L'istruzione Exit Sub ha lo scopo di uscire dalla routine senza eseguire le istruzioni seguenti; notate che, se invece di una routine fossimo stati in una funzione, l'istruzione per uscire da essa sarebbe stata **Exit Function**. All'interno del gestore degli errori viene usato l'oggetto Err, che contiene informazioni relative agli errori di run-time; in particolare, in questo esempio controlliamo il valore di Err.Number, che restituisce il numero dell'ultimo errore verificatosi. Alcune volte si usa la proprietà Err.Description, la quale contiene la descrizione dell'errore. Ora provate a premere F5 per eseguire il codice; osserverete che non comparirà più la finestra di errore di VB, ma la MessageBox che abbiamo definito noi:

Un'altra istruzione importante è l'istruzione **Resume**, che riprende l'esecuzione dopo il completamento di una routine di gestione degli errori. Solitamente è usata in due modi: Resume, riprende l'esecuzione dalla stessa istruzione che ha causato l'errore; Resume Next, riprende l'esecuzione dall'istruzione successiva a quella che ha provocato l'errore. Per esempio, sostituite questo pezzo di codice

```
GestoreErrori:
```

'Queste istruzioni vengono eseguite quando si verifica un errore.
If Err.Number = 11 Then
'Divisione per zero.
MsgBox "Si è verificato un errore di divisione per zero. Controllare i valori digitati e riprovare."
End If

Con

GestoreErrori:
'Queste istruzioni vengono eseguite quando si verifica un errore.
If Err.Number = 11 Then
'Divisione per zero.
MsgBox "Si è verificato un errore di divisione per zero. Controllare i valori digitati e riprovare."
B = 1
Resume
End If

Il nuovo codice, dopo aver visualizzato il messaggio di errore, cambio il valore di B da 0 a 1 e infine, con un Resume, torna all'istruzione che aveva provocato l'errore, cioè C = A / B. Ora la divisione (che è diventata 5 / 1) viene eseguita correttamente, pertanto otterremo:

Adesso provate a sostituire l'istruzione **Resume** che abbiamo appena scritto con **Resume Next**: come abbiamo detto, con essa l'esecuzione salta all'istruzione successiva a quella che ha provocato l'errore, quindi C = B / A non verrà più eseguita, ma il programma passerà subito all'istruzione MsgBox "Il risultato della divisione è: " & C. Modificate il codice come suggerito e osservate il risultato.
L'istruzione **Resume Next** può anche essere usata insieme all'istruzione **On Error**: in questo modo si dice al programma che, ogni volta che si verifica un errore, il problema deve essere ignorato e l'esecuzione deve passare all'istruzione successiva. Per fare questo è sufficiente scrivere:

On Error Resume Next

Di solito è sconsigliabile seguire questa strada, perché, non gestendo gli errori, si possono verificare delle situazioni imprevedibili.
Nella prossima lezione metteremo in pratica quanto abbiamo detto aggiornando il visualizzatore di immagini che abbiamo realizzato nella lezione 13.

Lezione successiva
[Sommario]
TORNA SU

GUIDA AL VISUAL BASIC

LEZIONE 16: *Aggiorniamo il programma*

Dopo aver parlato, nella lezione precedente, della gestione degli errori in VB, possiamo aggiornare il visualizzatore di immagini che abbiamo realizzato nella lezione 13 , in modo da correggere quegli errori che si verificano, ad esempio, quando si seleziona un'unità non disponibile. Cominciamo proprio da questo punto. Avviate il programma e, nel *DriveListBox*, selezionate l'unità A: senza che nessun dischetto sia inserito: verrà generato l'errore di runtime 68, indicante una "periferica non disponibile". Dobbiamo aggiungere la gestione di questo errore nell'evento *Drive1_Change*, che viene eseguito quando si seleziona un'unità. Il codice originale

```
Private Sub Drive1_Change()
Dir1.Path = Drive1.Drive
End Sub
```

Va modificato in questo modo:

```
Private Sub Drive1_Change()
On Error GoTo GestoreErrori
Dir1.Path = Drive1.Drive
Exit Sub

GestoreErrori:

If Err.Number = 68 Then
'Periferica non disponibile.
MsgBox "Impossibile accedere all'unità specificata. Selezionare un'unità diversa e riprovare."
Drive1.Drive = Dir1.Path
End If
End Sub
```

Come abbiamo detto, l'errore che dobbiamo gestire è il 68, quindi, come prima cosa, nella routine di gestione utilizziamo la proprietà *Err.Number* per controllare il numero dell'errore: se il problema è dovuto ad una periferica non valida, visualizziamo un messaggio di errore e, con l'istruzione Drive1.Drive = Dir1.Path, facciamo sì che l'unità selezionata corrisponda a quella di cui si stanno visualizzando le cartelle.

Un'altra situazione di errore nel programma si può avere quando, nel *FileListBox*, si seleziona un file immagine non valido, ad esempio perché danneggiato oppure perché, nonostante l'estensione, è un file di un altro tipo. In questo caso, selezionando tale file, verrà generato l'errore di runtime 481, cioè "immagine non valida". Ora il codice da modificare è quello contenuto nell'evento *File1_Click*; ecco la nuova routine:

```
Private Sub File1_Click()
On Error GoTo GestoreErrori
Dim FileSelezionato As String

FileSelezionato = File1.Path & "\" & File1.FileName
Image1.Picture = LoadPicture(FileSelezionato)
```

Exit Sub

GestoreErrori:

If Err.Number = 481 Then
'Immagine non valida.
MsgBox "L'immagine selezionata non è valida."
End If
End Sub

Dopo quello che abbiamo detto il significato del codice appena scritto dovrebbe essere chiaro; come prima, in caso di errore, viene visualizzata una finestra di messaggio che informa sul problema.

C'è ancora un errore che dobbiamo gestire e che, a differenza di quelli visti finora, è un po' più difficile da identificare, perché si verifica solo in una determina situazione. Consideriamo la riga di codice

FileSelezionato = File1.Path & "\" & File1.FileName

Essa salva nella variaibile *FileSelezionato* il nome e il percorso completo del file; per fare questo, recupera il percorso del file, vi aggiunge un back-slash ("\") e, infine, inserisce il nome del file. Questa procedura funziona bene se la cartella in cui ci si trova non è quella principale del disco (quindi non è C:\, A:\ e così via). Se invece, siamo, ad esempio, in C:\, la proprietà File1.Path contiene già il back-slash, perciò, dopo l'esecuzione dell'istruzione sopra riportata la variabile FileSelezionato conterrà un valore di questo tipo: *C:\\File.bmp*. Stando così le cose, l'istruzione **Image1.Picture = LoadPicture (FileSelezionato)** causerà l'errore di runtime 76, cioè l'errore "impossibile trovare il percorso".

Adesso che abbiamo imparato a scrivere il codice per gestire gli errori potremmo ampliare la routine già scritta, ma in questo caso è più conveniente prevenire il problema, piuttosto che correggerlo una volta che si è verificato. L'idea che sta alla base della soluzione è questa: per evitare l'errore, dovremmo fare in modo che il back-slash venga aggiunto solo se non è già l'ultimo carattere della proprietà File1.Path. A tale scopo possiamo utilizzare la funzione **Right$(, n)**, che restituisce gli ultimi n caratteri della stringa specificata. Ad esempio, l'istruzione **Right$("Prova", 2)** restituisce la stringa "va". Vediamo ora il codice vero e proprio:

Private Sub File1_Click()
On Error GoTo GestoreErrori
Dim FileSelezionato As String

If Right$(File1.Path, 1) = "\" Then
FileSelezionato = File1.Path & File1.FileName
Else
FileSelezionato = File1.Path & "\" & File1.FileName
End If
Image1.Picture = LoadPicture(FileSelezionato)
Exit Sub

GestoreErrori:

If Err.Number = 481 Then
'Immagine non valida.
MsgBox "L'immagine selezionata non è valida."
End If
End Sub

Come abbiamo anticipato, la modifica introdotta controlla se la proprietà File1.Path comprende già il back-slash come ultimo carattere e solo in caso negativo lo aggiunge alla stringa *FileSelezionato*.

Potete scaricare la nuova versione del visualizzatore immagini che abbiamo realizzato in questa lezione facendo clic qui.

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

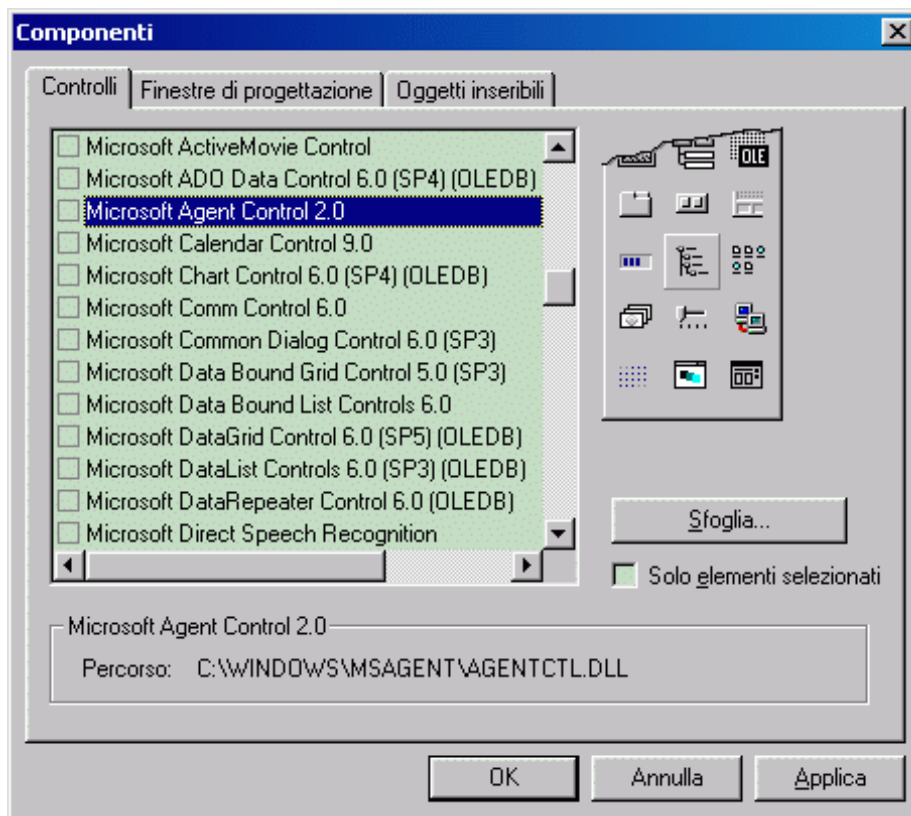
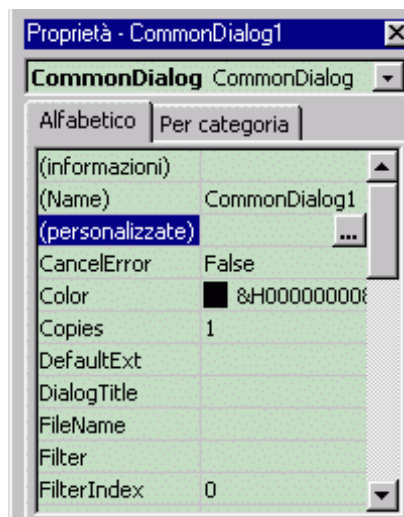
GUIDA AL VISUAL BASIC

LEZIONE 18: *Aggiungere un controllo OCX al progetto*

Finora abbiamo analizzato i controlli standard di Visual Basic, quelli sempre presenti nella Casella degli strumenti e che, quindi, possono essere utilizzati in tutti i programmi. In VB, come in tutti i linguaggi di programmazione, è possibile aggiungere nuovi controlli al progetto, rendendo così disponibili nuovi oggetti da inserire nel programma, nuove funzioni, ecc.

Tali controlli vengono chiamati in due modi: controlli **OCX** (o più brevemente OCX), dal momento che .ocx è l'estensione del file che li contiene, oppure controlli **ActiveX**. Una volta inseriti in un progetto, gli OCX possono essere trattati come un qualunque controllo standard, dal momento che mettono a disposizione proprietà, metodi ed eventi. Per inserire un controllo ActiveX in un progetto Visual Basic, è necessario fare clic sul comando **Componenti** (Components) del menu **Progetto** (Project); nella finestra di dialogo che si aprirà è possibile scegliere un ActiveX dall'elenco selezionando la relativa casella di controllo. Con un clic su **OK** o su **Applica**, esso verrà inserito nel progetto e sarà disponibile nella Casella degli strumenti insiemi agli altri componenti standard.

Proviamo subito ad utilizzare uno dei controlli OCX aggiuntivi forniti con VB, quello che permette di visualizzare le finestre di dialogo **Apri, Salva con nome, Carattere**, ecc. comuni a tutte le applicazioni Windows. Aprite la finestra **Componenti** seguendo le istruzioni riportate sopra e scorrete l'elenco fino a trovare *Microsoft Common Dialog Control*; selezionate questo controllo OCX con un clic sulla casella di spunta visibile a lato del nome e chiudete la finestra di dialogo con un clic sul pulsante **OK**.

Ora nella Casella degli strumenti apparirà una nuova **icona**, corrispondente ad un nuovo controllo che è possibile utilizzare nella propria applicazione. Inseritelo nel form come avete sempre fatto con tutti i controlli standard. Nella finestra delle Proprietà del controllo potete notarne una chiamata (personalizzate): selezionatela e fate clic sul pulsante con i tre puntini visibile a destra: si aprirà una nuova finestra in cui è possibile modificare le proprietà del controllo. Ora premete **Annulla**. Impostate la proprietà CancelError su True, in modo che venga generato un errore di runtime se l'utente preme il tasto **Annulla** nelle finestre di dialogo (ci servirà per l'esempio che andremo a realizzare).

Per provare il controllo vogliamo richiamare la finestra di dialogo **Apri** per selezionare un file. In questo contesto ci interessa solo analizzare il comportamento dell'OCX, quindi per richiamarlo andrà benissimo un semplice pulsante. Inserite dunque un CommandButton nel form e modificate la sua Caption, ad esempio, in "Apri file". Il Microsoft Common Dialog Control dispone di cinque metodi fondamentali, *ShowOpen, ShowSave, ShowFont, ShowColor e ShowPrinter*, per visualizzare le finestre di dialogo rispettivamente per l'apertura di un file, per il salvataggio, per la selezione del carattere, per la selezione del colore, per l'impostazione della stampante.

Ad essi va aggiunto il metodo ShowHelp, che visualizza la Guida in linea. Proviamo, ad esempio, ad utilizzare il metodo ShowOpen:

```
Private Sub Command1_Click()
'Visualizza la finestra di dialogo per l'apertura di un file.
CommonDialog1.ShowOpen
'Visualizza una MessageBox contenente il nome del file selezionato.
MsgBox "Il file selezionato è: " & CommonDialog1.FileName
End Sub
```

In questo codice, come detto, viene utilizzato il metodo ShowOpen, dopodiché

si usa la proprietà FileName per recuperare il nome completo del file selezionato.

Dopo aver fatto qualche prova per verificare il funzionamento di questa routine, provate ad aprire la finestra ma, invece di selezionare un file, premete il tasto **Annulla**; se come suggerito avete impostato la proprietà CancelError su True, a questo punto verrà generato l'errore di runtime 32755. Ecco quindi un'altra situazione in cui dobbiamo prevedere una gestione degli errori. Dopo quello che si è detto nelle precedenti lezioni l'aggiunta non dovrebbe essere difficile da realizzare, ecco come dovrà apparire il codice dopo la modifica:

```
Private Sub Command1_Click()
On Error GoTo GestoreErrori
'Visualizza la finestra di dialogo per l'apertura di un file.
CommonDialog1.ShowOpen
'Visualizza una MessageBox contenente il nome del file selezionato.
MsgBox "Il file selezionato è: " & CommonDialog1.FileName
Exit Sub

GestoreErrori:

If Err.Number = 32755 Then
'E' stato premuto Annulla.
MsgBox "E' stato premuto il pulsante 'Annulla'."
End If
End Sub
```

L'esempio appena realizzato è disponibile per il download facendo clic qui.
Non è possibile in questa sede analizzare più approfonditamente il funzionamento del controllo; per maggiori informazioni su questo e, in generale, su tutti gli altri controlli OCX utilizzabili con VB, si raccomanda di consultare la Guida in linea che li accompagna.

GUIDA AL VISUAL BASIC

LEZIONE 18: ***Un'agenda elettronica con VB: l'interfaccia***

A questo punto abbiamo tutti gli strumenti necessari per iniziare a lavorare con VB con un certo profitto. Per fissare quanto è stato detto vogliamo ora realizzare un vero e proprio programma, più precisamente un'agenda elettronica che contiene i nomi e gli indirizzi dei nostri amici. Ne approfitteremo per introdurre nuove funzioni e nuovi comandi che non abbiamo ancora analizzato.

I dati della rubrica saranno salvati in un database di Microsoft Access, a cui accederemo dal nostro programma utilizzando il controllo Data, che fa parte dei controlli standard di Visual Basic; esso offre tutte le funzioni necessarie per visualizzare le informazioni contenute in un database, quindi ci permetterà di visualizzare i dati, modificarli, eliminarli, fare una ricerca. Potete scaricare una copia del database che utilizzeremo facendo clic qui. Sono stati previsti 4 tipi di informazione: *Nome, Indirizzo, Telefono e e-Mail*.

Per poter utilizzare il database fornito nell'esempio ed evitare il messaggio "Impossibile trovare ISAM installabile", potrebbe esser necessario installare alcuni moduli supplementari ISAM che permettono l'utilizzo di banche dati all'interno di Visual Basic. Per ottenere questi moduli è consigliabile aggiornare il Microsoft Jet Engine all'ultimo Service Pack fornito da Microsoft.

Non perdiamo altro tempo e iniziamo subito a costruire il programma; quando introdurremo nuove funzioni o nuovi concetti ci fermeremo ad analizzare il codice.

Per prima cosa modifichiamo la Caption del form in "Agenda elettronica", poi impostiamo la proprietà BorderStyle su 1 - Fixed Single (abbiamo diffusamente parlato del form nella lezione 8). Ora scorriamo la finestre delle

Proprietà fino a trovare MinButton: impostiamola su **True**; in questo modo nella barra delle applicazioni del form verrà visualizzato il pulsante di riduzione a icona. Possiamo anche impostare un'icona che contraddistinguerà la nostra applicazione (anche questo aspetto è stato trattato nella lezione 8); quella utilizzata nell'esempio è disponibile facendo clic qui. L'ultima proprietà del form che vogliamo modificare è StartUpPosition: impostandola su 2 - **CenterScreen** la finestra verrà sempre visualizzata al centro dello schermo.

Ora modifichiamo le **dimensioni** del form, dal momento che la nostra applicazione conterrà un discreto numero di oggetti: per fare questo è sufficiente fare clic con il tasto sinistro del mouse sul quadratino visualizzato nell'angolo in basso a destra del form e, tenendo il pulsante premuto, trascinare il mouse per impostare la nuova dimensione, che sarà fissata una volta rilasciato il tasto stesso.

Ora aggiungiamo gli altri **elementi** dell'interfaccia. Ci servono 4 TextBox, una per ogni tipo di informazione che vogliamo visualizzare; per ognuna di queste aggiungeremo una Label descrittiva. Abbiamo poi bisogno di *7 CommandButton*: i primi quattro saranno utilizzati per trovare, aggiungere, modificare ed eliminare i dati, uno servirà per aggiornare i dati visualizzati, un altro per salvare le modifiche effettuate, mentre l'ultimo consentirà di uscire dal programma. Naturalmente ci serve anche il controllo *Data*, che ci permetterà di accedere ai dati contenuti nel database. Possiamo infine inserire un'etichetta descrittiva dell'applicazione, affiancata da un'icona.

In conclusione, il form dovrebbe risultare simile a quello riprodotto sopra e che può essere scaricato facendo clic qui. Come di consueto, la tabella seguente elenca i controlli che vanno inseriti nel form con i valori delle rispettive proprietà:

Nome controllo (tipo)	**Proprietà**	**Valore**
Image1 (*Image*)	Picture	Questa immagine
Label1 (*Label*)	AutoSize	True
	Caption	Agenda elettronica
	Font	Arial, 14 punti
Data1 (*Data*)	Caption	Agenda
Label2 (*Label*)	Caption	Nome:
	AutoSize	True
Text1 (*TextBox*)	Text	(vuoto)
Label3 (Label)	Caption	Indirizzo:
	AutoSize	True
Text2 (*TextBox*)	Text	(vuoto)
Label4 (Label)	Caption	Telefono:

	AutoSize	True
Text3 (*TextBox*)	Text	(vuoto)
Label5 (*Label*)	Caption	e-Mail
	AutoSize	True
Command1 (*CommandButton*)	Caption	Trova
Command2 (*CommandButton*)	Caption	Aggiungi
Command3 (*CommandButton*)	Caption	Modifica
Command4 (*CommandButton*)	Caption	Elimina
Command5 (CommandButton)	Caption	Aggiorna
Command6 (CommandButton)	Caption	Salva
	Visibile	False
Command7 (CommandButton)	Caption	Esci

Nella prossima lezione imposteremo le proprietà del controllo Data per fare in modo che nelle varie TextBox vengano visualizzati i dati prelevati dal database.

Lezione successiva

[Sommario]

TORNA SU

GUIDA AL VISUAL BASIC

LEZIONE 19: *Attiviamo la navigazione nel database*

Il passo successivo nella realizzazione del programma è la configurazione del controllo *Data* e la sua sincronizzazione con le varie *TextBox*, in modo che queste ultime visualizzino i dati prelevati dal database. Come vedremo tra breve, si tratta di un'operazione semplicissima, che richiede la stesura di una sola riga di codice, a dire il vero neanche strettamente necessaria, ma utile per garantire il buon funzionamento del programma in ogni situazione.

Per prima cosa dobbiamo configurare il controllo Data in modo che vada a recuperare le informazioni dal database che abbiamo creato; per fare questo è necessario modificare la proprietà *DatabaseName*. Facendo clic sul pulsante con i tre puntini visibile a destra della proprietà suddetta comparirà la finestra **Nome database**, in cui dobbiamo selezionare il file MDB che vogliamo utilizzare nella nostra applicazione. Una volta effettuata, confermiamo la scelta con un clic su **Apri**.

DatabaseName verrà aggiornata con il nome e il percorso completo del file appena selezionato. Questo ultimo particolare ci fa intuire per quale motivo dobbiamo ora scrivere una semplice riga di codice: dal momento che viene utilizzato il percorso completo del database, se copiamo il programma in un'altra cartella esso non funzionerà più, poiché tenterà di aprire un file non più esistente. Per risolvere il problema è sufficiente aggiornare la proprietà *DatabaseName* all'interno dell'evento Form_Load:

```
Private Sub Form_Load()
Data1.DatabaseName = App.Path & "\Agenda.mdb"
End Sub
```

Dobbiamo ancora modificare una proprietà del controllo Data, più precisamente la proprietà *RecordSource*, che consente di impostare il recordset del database che si intende utilizzare. Selezioniamo **Rubrica**. La proprietà RecordsetType, infine, deve essere impostata su 0 - Table.

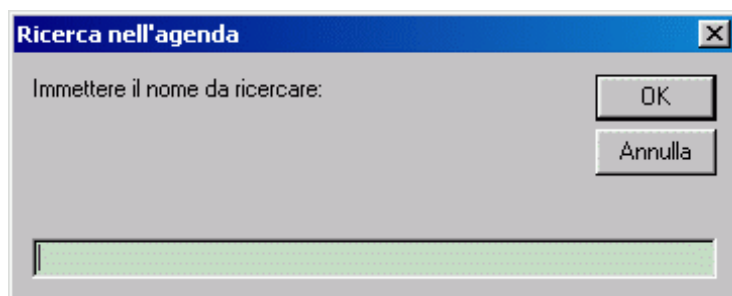
Ora non ci resta che collegare le varie TextBox al controllo Data. Selezioniamo la prima casella di testo e selezioniamo, per la proprietà DataSource, il controllo Data1. A questo punto, facendo clic sul pulsante a lato della proprietà DataField comparirà l'elenco dei campi del database: selezioniamo **Nome**, per fare in modo che nella prima casella di testo venga visualizzato il nome del contatto. Ripetiamo questi passaggi per le altre TextBox: la proprietà *DataSource* deve sempre essere impostata su Data1, mentre la proprietà DataField deve essere, rispettivamente, **Indirizzo** (Text2), **Telefono** (Text3) e **E-Mail** (Text4). Tutto qui: ora il programma può già essere utilizzato per la consultazione del database. Proviamo subito premendo il tasto F5. Verrà visualizzato il primo record del database; agendo sui pulsanti a lato del controllo Data sarà possibile spostarsi in aventi o indietro: le varie caselle di testo verranno automaticamente aggiornate in modo da visualizzare i dati corretti.
Aggiungiamo infine due righe di codice, una che verrà eseguita quando si preme il pulsante **Aggiorna**, l'altra alla pressione del tasto **Esci**. In questo secondo caso, vogliamo semplicemente chiudere il programma, quindi dobbiamo scrivere:

```
Private Sub Command7_Click()
'Esce dal programma.
Unload Me
End Sub
```

Premendo il tasto **Aggiorna**, invece, vogliamo che il controllo Data venga aggiornato, così da rendere effettive le eventuali modifiche (potrebbe tornarci utile nelle prossime Lezioni). Si può raggiungere lo scopo semplicemente richiamando il metodo Refresh del controllo:

```
Private Sub Command5_Click()
'Aggiorna il controllo.
Data1.Refresh
End Sub
```

Fate clic qui per scaricare il programma al punto in cui siamo arrivati.

Lezione successiva
[Sommario]
TORNA SU

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 20: ***La funzione di ricerca***

Dopo aver configurato il programma per consentire la navigazione nel database, in questa Lezione vediamo come aggiungere la funzione di ricerca. Per definire tale operazione (ma anche le altre che vedremo nelle Lezioni successive) ci serviremo dell'oggetto *Recordset*, accessibile a partire dal controllo Data; esso dispone di metodi e di proprietà che consentono di lavorare con i dati prelevati dal database.

La prima funzione che vogliamo definire è la ricerca di un nominativo: quando si preme il pulsante **Trova**, deve apparire una finestra che chiede di immettere il nome da cercare e, successivamente, deve essere eseguita la ricerca vera propria. Ecco il codice che svolge tale compito:

```
Private Sub Command1_Click()
'Ricerca un nome all'interno dell'agenda.
Dim NomeDaCercare As String
NomeDaCercare = InputBox$("Immettere il nome da ricercare:", "Ricerca
nell'agenda")
If NomeDaCercare <> "" Then
'Esegue la ricerca solo se è stato immesso un nome.
Data1.Recordset.Index = "Nome"
Data1.Recordset.Seek "=", NomeDaCercare
If Data1.Recordset.NoMatch Then
Data1.Recordset.MoveFirst
Data1.Refresh
'Il nome cercato non è stato trovato.
MsgBox "Nome non trovato.", vbInformation, Me.Caption
End If
End If
End Sub
```

Cerchiamo di capire la logica di funzionamento di questa routine. Innanzi tutto viene richiesto all'utente di immettere il nome da cercare; per fare questo si usa la funzione **InputBox**, che visualizza una finestra invitando ad immettere il valore richiesto; tale funzione accetta 7 parametri, ma di questi solo il primo è obbligatorio e rappresenta il messaggio visualizzato nella finestra stessa. Il secondo parametro consente di impostare il titolo della finestra; se non viene specificato, verrà utilizzato il titolo della finestra che richiama la funzione. Il terzo parametro indica il valore iniziale (predefinito) visualizzato nella casella di testo. Per ulteriori informazioni sulla funzione **InputBox** si consiglia di consultare la Guida in linea di Visual Basic. La chiamata della funzione **InputBox** nella nostra applicazione visualizzerà la finestra riprodotta a lato. Il valore immesso in tale finestra viene salvato nella variabile NomeDaCercare.

A questo punto, se è stato specificato un nome da ricercare, viene impostata la proprietà Index dell'oggetto **Recordset**, con la quale si stabilisce quale campo utilizzare per la ricerca. Subito dopo, il metodo Seek ricerca il nome all'interno del database: come primo parametro abbiamo utilizzato "=" perché vogliamo trovare esattamente il nome specificato.

Eseguita la ricerca, ci preoccupiamo di controllare se essa ha avuto esito positivo oppure no, ovvero se non è stato trovato nessun record che corrisponde ai criteri digitati; allo scopo, controlliamo il valore della proprietà *NoMatch*, che restituisce True se la ricerca ha avuto esito negativo: in questo caso ci spostiamo nel primo record utilizzando il metodo *MoveFirst*, poi visualizziamo una finestra di messaggio informando l'utente che il nome specificato non è stato trovato. Potete notare che, in questo caso, per la funzione MsgBox utilizziamo nuovi argomenti, oltre al consueto testo del messaggio. Il secondo parametro (opzionale, come il terzo) è un'espressione numerica che indica il numero e il tipo di pulsanti da visualizzare, lo stile di icona da utilizzare, il pulsante predefinito e la modalità della finestra; se è omesso, il valore predefinito è 0. L'espressione numerica può essere sostituita, come nel nostro programma, da una costante definita da VB: in questo caso, vbInformation indica che nella finestra deve essere visualizzata l'icona che contraddistingue un messaggio di informazione. Il terzo parametro specifica il titolo della finestra. Per maggiori informazioni, come sempre potete fare riferimento alla Guida in linea.

Tornando al discorso precedente, se viene trovato un record che corrisponde ai criteri digitati esso diventa automaticamente il record corrente, quindi viene visualizzato nella finestra principale del programma.

Potete scaricare il programma con la nuova funzione facendo clic qui.

Lezione successiva
[Sommario]

TORNA SU

Home page
Guida Base
Guida al Java
Guida al C
Guida al C++
Guida al Delphi
Guida a VB .NET
Guida al Visual Basic
Guida al Python
Guida al'UML
Forum di discussione
HTML.it

GUIDA AL VISUAL BASIC

LEZIONE 21: ***La funzione di modifica***

Il passo successivo nella realizzazione della nostra applicazione è la definizione delle funzioni di modifica e di aggiunta di informazioni nell'agenda. Vediamo prima la modifica. Vogliamo fare in modo che, quando l'utente preme il tasto **Modifica**, la Caption del pulsante cambi in **Annulla** e venga visualizzato il pulsante Salva; premendo quest'ultimo le modifiche devono essere salvate nel database, mentre facendo clic sul pulsante **Annulla** i cambiamenti devono essere annullati. Ancora una volta, utilizzando i metodi dell'oggetto Recordset queste operazioni possono essere svolte molto semplicemente. Il codice che deve essere eseguito alla pressione del tasto **Modifica** è il seguente:

```
Private Sub Command3_Click()
If Command3.Caption = "Modifica" Then
'Attiva la funzione di modifica.
Data1.Recordset.Edit
Command3.Caption = "Annulla"
Command6.Visible = True
Else
'Annulla le modifiche.
Data1.Recordset.CancelUpdate
Command6.Visible = False
Command3.Caption = "Modifica"
End If
End Sub
```

Innanzi tutto viene controllata la *Caption* del pulsante: se è "Modifica", richiama il metodo Edit per attivare la funzione di modifica; fatto questo, imposta la Caption su "Annulla" e mostra il pulsante **Salva**. Viceversa, se la Caption è "Annulla", annulla le eventuali modifiche utilizzando il metodo CancelUpdate, nasconde il pulsante **Salva** e reimposta la Caption su "Modifica". Ora dobbiamo scrivere il codice che verrà eseguito alla pressione del tasto **Salva**:

```
Private Sub Command6_Click()
'Salva le modifiche.
Data1.Recordset.Update
Command6.Visible = False
Command3.Caption = "Modifica"
End Sub
```

Il metodo Update salva le modifiche nel database; successivamente il pulsante **Salva** viene nascosta e Caption del pulsante **Modifica** viene reimpostata.

A questo punto abbiamo scritto il codice necessario alla modifica dei dati; tuttavia c'è ancora una situazione che non abbiamo previsto: è possibile che l'utente faccia clic sulle frecce di navigazione del controllo Data durante un'operazione di modifica; in questo caso vogliamo che compaia una finestra per confermare il salvataggio. L'evento che utilizzeremo è l'evento Validate del controllo Data, che viene generato, tra gli altri casi, prima che un record diventi il record corrente e prima del metodo *Update*. Ecco il codice:

```
Private Sub Data1_Validate(Action As Integer, Save As Integer)
Dim Risposta As Integer
If Save = True Then
'E' stato modificato un contatto.
Risposta = MsgBox("Salvare le modifiche apportate?", vbQuestion + vbYesNo,
Me.Caption)
If Risposta = vbNo Then
'Se si seleziona no, annulla le modifiche.
'Per questo, la proprietà DataChanged delle TextBox viene impostata su
False.
Text1.DataChanged = False
Text2.DataChanged = False
Text3.DataChanged = False
Text4.DataChanged = False
End If
End If
'Nasconde il pulsante "Salva".
Command6.Visible = False
Command3.Caption = "Modifica"
End Sub
```

Analizziamo questa routine. L'evento *Validate* ha due parametri, Action e Save: il primo identifica l'operazione che ha generato l'evento (consultate la Guida in linea per maggiori informazioni), mentre il secondo contiene un valore booleano che determina se i dati sono stati modificati. Innanzi tutto controlliamo se il valore di Save è True, cioè se è stata fatta qualche modifica; in tal caso, visualizziamo una finestra che chiede all'utente di confermare il salvataggio.

Per fare questo utilizziamo nuovamente la MsgBox, ma ora in un modo diverso; il secondo argomento della funzione è vbQuestion + vbYesNo, ovvero è la somma di due costanti: la prima indica che vogliamo visualizzare l'icona di richiesta (un punto interrogativo), mentre la seconda fa in modo che nella finestra vengano visualizzati i pulsanti **Sì** e **No**.

A questo punto sorge una domanda: come si recupera la scelta dell'utente, cioè come si può sapere su quale dei due pulsanti è stato fatto clic? Abbiamo detto più volte che la MsgBox è una funzione e, quindi, come tale, restituisce un valore (abbiamo parlato nelle funzioni nella lezione 3); il valore restituito dalla funzione è memorizzato nella variabile Risposta e, nel nostro programma, può essere vbYes (l'utente ha selezionato **Sì**) oppure vbNo (è stato selezionato **No**). In quest'ultimo caso le modifiche vengono annullate ponendo la proprietà DataChanged delle caselle di testo su False (per maggiori informazioni, si consiglia come sempre di consultare la Guida in linea). Le ultime due istruzioni dell'evento Validate nascondono il pulsante **Salva** e reimpostano la Caption del pulsante **Modifica**.

Potete scaricare il programma con le funzioni di ricerca e di modifica dei dati facendo clic qui.

[Sommario]

TORNA SU

GUIDA AL VISUAL BASIC

LEZIONE 22: ***L'aggiunta di dati***

L'aggiunta di informazioni nel database è un'operazione per certi versi analoga alla modifica dei dati; anche il codice che andremo a scrivere sarà simile. Vediamo subito il codice da scrivere nell'evento Click del pulsante **Aggiungi**:

```
Private Sub Command2_Click()
If Command2.Caption = "Aggiungi" Then
'Attiva la funzione di aggiunta.
Data1.Recordset.AddNew
Command2.Caption = "Annulla"
Command6.Visible = True
Else
'Annulla le modifiche.
Data1.Recordset.CancelUpdate
Command6.Visible = False
Command2.Caption = "Aggiungi"
End If
End Sub
```

Come si vede, anche in questo caso per il salvataggio dei dati vogliamo utilizzare il pulsante **Salva** (Command6). Dobbiamo allora fare una piccola aggiunta nell'evento *Command6_Click*:

```
Private Sub Command6_Click()
'Salva le modifiche.
Data1.Recordset.Update
Command6.Visible = False
Command2.Caption = "Aggiungi"
Command3.Caption = "Modifica"
End Sub
```

L'istruzione che abbiamo aggiunto, Command2.Caption = "Aggiungi", ha lo scopo di reimpostare la Caption del pulsante **Aggiungi**, in modo analogo a quanto abbiamo nella lezione precedente per il tasto **Modifica**. Ma allora dobbiamo ricordarci di reimpostare la Caption anche nell'evento Validate del controllo Data, che quindi diventerà:

```
Private Sub Data1_Validate(Action As Integer, Save As Integer)
Dim Risposta As Integer
If Save = True Then
'E' stato modificato un contatto.
Risposta = MsgBox("Salvare le modifiche apportate?", vbQuestion + vbYesNo,
Me.Caption)
If Risposta = vbNo Then
'Se si seleziona no, annulla le modifiche.
'Per questo, la proprietà DataChanged delle TextBox viene impostata su
False.
Text1.DataChanged = False
Text2.DataChanged = False
Text3.DataChanged = False
Text4.DataChanged = False
```

```
End If
End If
'Nasconde il pulsante "Salva".
Command6.Visible = False
Command2.Caption = "Aggiungi" 'ISTRUZIONE AGGIUNTA.
Command3.Caption = "Modifica"
End Sub
```

Abbiamo finito: ora è possibile aggiungere dati alla nostra agenda. Il programma aggiornato con la nuova funzione è disponibile per il download facendo clic qui.

L e z i o n e   s u c c e s s i v a

[S o m m a r i o]

TORNA SU

GUIDA AL VISUAL BASIC

LEZIONE 23: ***L'eliminazione dei dati***

L'ultima funzione che dobbiamo aggiungere al nostro programma è quella che consente di eliminare i dati precedentemente inseriti. Anche in questo caso vogliamo che, prima di eseguire l'operazione, una *MessageBox* ci chieda una conferma. Ecco dunque quello che dobbiamo scrivere nell'evento Click del pulsante **Elimina** (Command4_Click):

```
Private Sub Command4_Click()
Dim Risposta As Integer
'Chiede conferma prima di procedere con l'eliminazione.
Risposta = MsgBox("Eliminare i dati correnti?", vbQuestion + vbYesNo, Me.
Caption)
If Risposta = vbYes Then
'Elimina i dati.
Data1.Recordset.Delete
'Si sposta nel record precedente.
Data1.Recordset.MovePrevious
End If
End Sub
```

Se l'utente risponde **Sì** alla domanda che gli viene posta (Risposta = vbYes), cancelliamo il record attualmente visualizzato utilizzando il metodo **Delete** dell'oggetto **Recordset**. Fatto questo, con l'istruzione Data1.Recordset. MovePrevious ci si sposta nel record precedente.
Abbiamo così terminato di scrivere il nostro programma; ora tutte le funzioni sono disponibili e correttamente funzionanti. Potete scaricare l'applicazione completa facendo clic qui.

Si conclude con questa Lezione il nostro corso su Visual Basic. Siamo partiti da un'analisi delle caratteristiche peculiari di questo ambiente di sviluppo, per poi presentare i principali strumenti che VB ci mette a disposizione per realizzare un'applicazione. Infine, partendo dal bagaglio di conoscenze acquisite, abbiamo realizzato un programma completo e ne abbiamo approfittato per introdurre nuovi concetti e nuove funzioni che non erano ancora state analizzate.